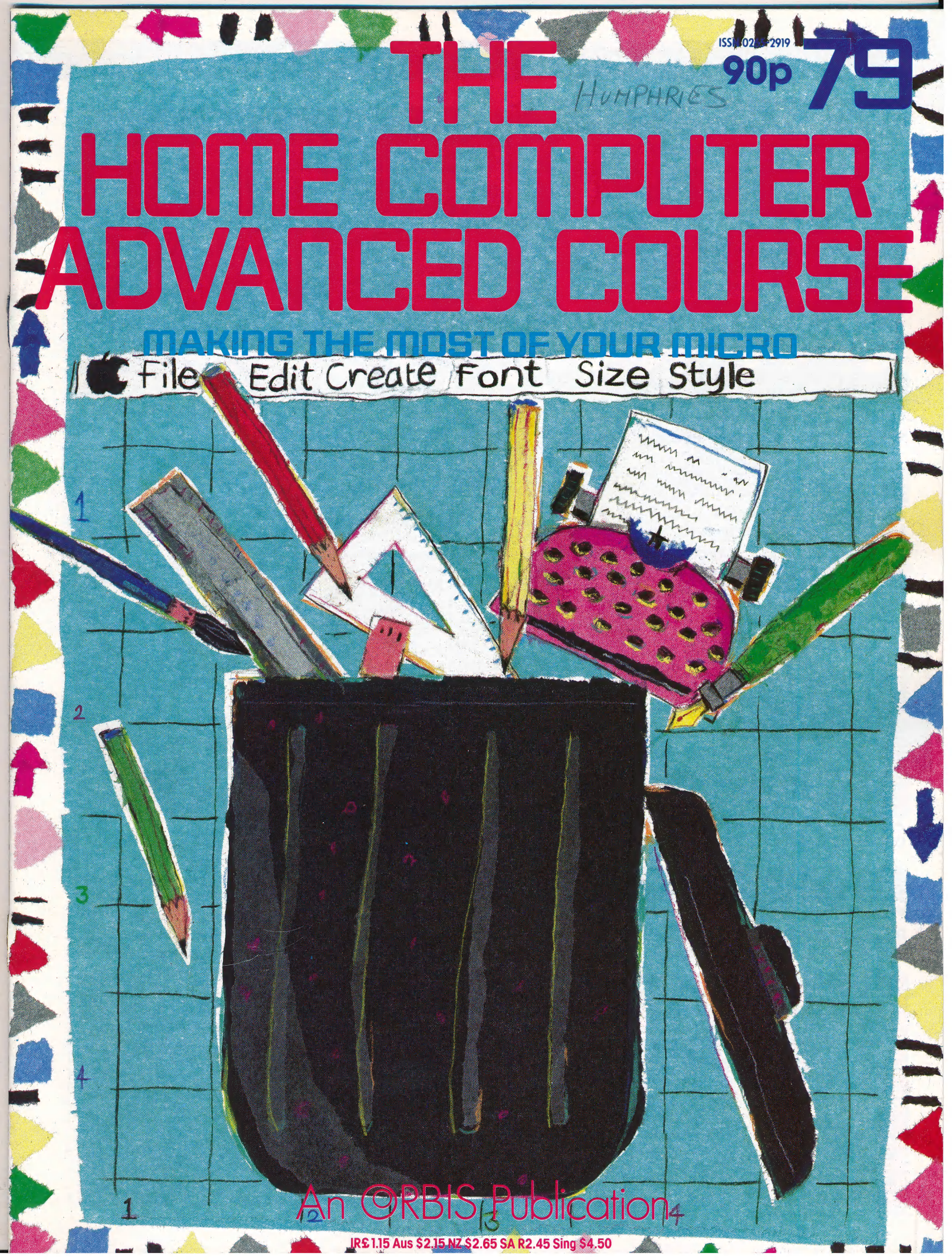
ISSN 0262-2919

90p 79

THE HOME COMPUTER ADVANCED COURSE

HUMPHRIES

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

UK £1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

NEW TECHNOLOGY SHOCK! The production of newspapers and magazines is increasingly coming under the control of computerised systems 1561

HARDWARE

BIG BROTHER The IBM PC/AT uses a process called 'virtual memory management', which will undoubtedly be implemented on home micros 1569

SOFTWARE

OBJECT LESSON Object manipulation is a vital feature of interactive character games 1575

COMPUTER SCIENCE

PLAY ON WORDS We look at how FORTH handles string manipulation 1567

JARGON

FROM SEARCH ALGORITHM TO SEQUENTIAL ACCESS A weekly glossary of computing terms 1580

PROGRAMMING PROJECTS

A CLEAN SHEET We look at what a spreadsheet program is, and begin designing our own 1564

MACHINE CODE

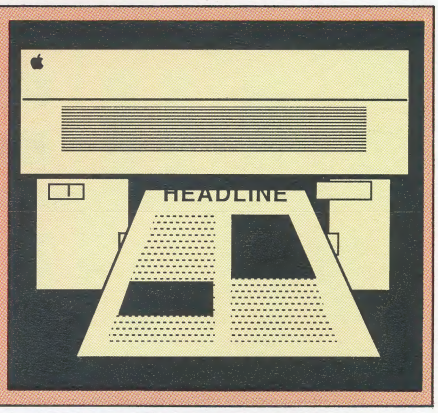
TICKED OFF A look at the Amstrad operating system's timing techniques 1578

WORKSHOP

WHEN THE CHIPS ARE DOWN We finish constructing our digital multimeter and show you how to test the connections 1572

Next Week

- **A look at the Pioneer PX-7, an MSX computer that forms the heart of an interactive video system.**
- **We show how a magazine can be produced — from manuscript to printed page — on a single micro system.**
- **Following on from this week's look at the Amstrad's use of interrupts, we delve into the intricacies of the OS 'events'.**



QUIZ

1) What is a 'direct entry' printing system?
2) Why can't the IBM PC read the disks from the PC/AT?
3) What can be put in a spreadsheet cell?
4) What is a 'p-type' semiconductor?

Answers To Last Week's Quiz

1) The operating system of the 520ST is the Tramiel Operating System (TOS) and is a derivative of CP/M-68

2) An actuator provides the appropriate response to a signal in a computerised process control system.

3) A character transfer moves data from a disk or cassette into a buffer, which can then be transferred elsewhere within memory. Block transfers, however, move the information directly from the medium to memory.

Coming Up

- A guide to the 68000 Assembly language used on the Sinclair QL.
- A look at GEM, Digital Research's 'graphics environment manager' created for 16-bit machines such as the new Atari 520 ST.

THE SNOWBALL EFFECT We review an adventure game for the Amstrad computers INSIDE BACK COVER

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editors** Jon Kaye, **Contributors** Chris Honey, Chris Laing, Dougie Bern, Jon Kaye, Steve Cooke, Marcus Jeffery, Steve Colwill, Steve Malone, Steven Vickers, David Mudd; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985; Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited. Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER ILLUSTRATION BY SALLY DAVIES

NEW TECHNOLOGY SHOCK

Many Fleet Street newspapers still use a machine known as a linotype to set the type from which the newspaper pages are printed. Invented in 1886, the linotype casts complete lines of type in molten metal from copy entered at the keyboard. Computerised typesetting is increasingly being used by hard-pressed provincial newspapers in order to keep costs down. These new machines allow copy to be loaded electronically, and moulded plastic printing plates called 'stereotypes' are produced

We begin a two-part series that focuses on the applications of electronic technology in the publishing industry. By outlining the current methods of production and understanding the problems involved, we can gain a greater awareness of the dramatic impact computers are making on magazine production.

It has become something of a cliché to suggest that computer technology is radically changing the way people live and work. Nevertheless, the introduction of new computer-based methods is sweeping away many established practices and skills. One area that particularly illustrates this change in working patterns is publishing, in which computer technology is being introduced into an industry that has not, in many cases, updated its methods and machinery since the 1930s. Additionally, the industry's labour force is organised into traditionally strong unions. Thus the introduction of computers is likely to be a traumatic experience for those working within the business.

In this short series, we'll outline the changes that are taking place in the publishing industry and where current trends are likely to lead. First, let's look at the new technology and see how it is applied to each stage of the publishing process.

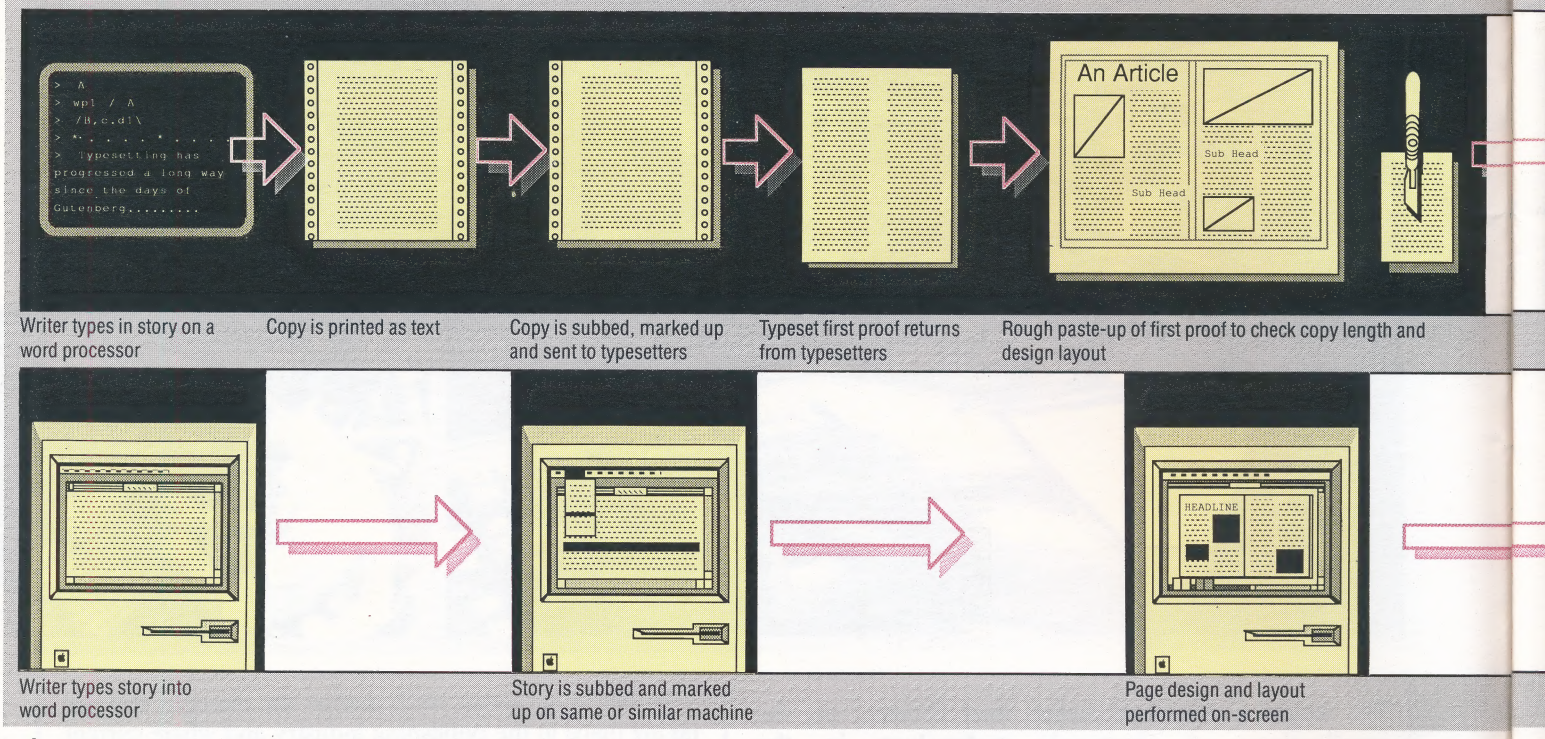
The most obvious place to start is at the written word. Most writers have now replaced their mechanical (perhaps electric) typewriters with microcomputers, which have allowed them to write and correct their own text, increase their work-rate and produce neat, inexpensive copy for publishers. You'll no doubt be familiar with word processing programs on disk and tape or on special ROMs that can be installed in your micro, which provide it with word processing capabilities. In fact, after game-playing, word processing is the most widely used application of home micros.

Whether you simply wish to send a newsletter to local club members or write your first Mills and Boon romance, it's likely that the humble home micro will be a powerful enough tool for your purposes. Copy can be checked and corrected on screen, and margin widths, automatic page

numbering, page headings and footings can all be set up prior to printing to produce a single finished draft of the text. To make life even easier, many word processing packages have spelling checkers that scan through the text and point out any unrecognised words for correction. If the text is to form an article in a magazine or newspaper, the production of neatly typed copy is only the first stage in the lengthy process of producing a final printed version.

In most current magazine production methods, the traditional systems still prevail. The author of the piece is normally required by the publication to submit the article typed (usually by a normal typewriter or by a word processor/printer arrangement). The article is then edited by editorial staff on the publication. The editor reads and corrects the *content* of the piece, normally by writing on the piece by hand or, where major alterations are required, by

The Changing Face of Printing


Typesetting has progressed a long way since the days of Gutenberg.....
An Article
Sub Head
Sub Head
Writer types in story on a word processor
Copy is printed as text
Copy is subbed, marked up and sent to typesetters
Typeset first proof returns from typesetters
Rough paste-up of first proof to check copy length and design layout
Writer types story into word processor
Story is subbed and marked up on same or similar machine
HEADLINE
Page design and layout performed on-screen

producing typewritten inserts. It's then passed on to a sub editor who corrects spelling mistakes and grammatical errors (again normally by making handwritten corrections on the piece), rewrites portions to conform with the publication's style, and marks up the article for typesetters. The process of 'marking up' is essentially a way of telling the typesetter the typeface, margin widths and justification to use and is again normally written by hand on the author's original copy. On small publications, the editor and sub editor may well be one and the same person.

When the article returns from the typesetters, it is hoped that it is near its final printed form, although it is likely that the typesetters will have made transcription errors in copying the original marked-up text onto the typesetting machines. The printed article is known at this stage as the first galley, or first proof, and must be carefully re-read by the sub editor, corrected and returned to the typesetters who will produce a revised version of the article — the second proof, naturally enough.

At the first proof stage, another person enters the production process. It is the job of the designer to take the first printed proof of the text and design a page layout: the way that the final printed form will look. Of course, most magazines contain not only text, but also photographs and diagrams together with appropriate captions. The designer must juggle all the components that will constitute the final content of the page in order to fit them together in an attractive and sensible way.

The most common way of doing this is to make a 'mock-up' of the page and fit the various components like pieces in a jigsaw puzzle. The first proof text may also be physically laid out on the page prototype at this stage, although some designers merely take measurements of the text length in order to determine the space it will occupy on the finished page.

It is often the case during this process that the text for the article is found to be either too long or too short to fit into the space allotted to it. In this event, the editorial and sub editorial staff must either make sensible cuts to the piece or, if it is too short, add text to bring the article up to the correct length. Cuts or additions to the text are included in the first proof corrections to the typesetters, so that by the time the second proof is returned, the length of the text is correct. Finally, the page is assembled with diagrams, photographs and text according to the previously worked-out design, and the finished page can be sent off for printing.

From this brief description of current practices in magazine production, we can see that the production process that turns a contributor's copy into a finished article in the magazine involves many, potentially time-wasting, stages. The system just described is in fact a kind of hybrid system in which single stand-alone computers are used in isolation during parts of the process.

The article itself is likely to be written on a word processor, but the typesetting process as well is now likely to be computerised, using a machine not unlike a normal word processor but with extra commands included to allow

the selection of different typefaces and so on. Looking at such a system, it doesn't take a professional systems analyst to suggest possible improvements.

First, imagine the magazine editorial office having a compatible word processing system to that used by the author. The author would not need to produce printed pages but merely hand over a disk or tape with the article recorded on it magnetically, or even transmit the article electronically using the telephone network and a couple of modems. All editing and sub editing of the article could then be done on screen. Even if the process then reverted to the more traditional method at this stage, by printing the edited version of the article onto paper to be sent to the typesetters, the system has already been somewhat streamlined.

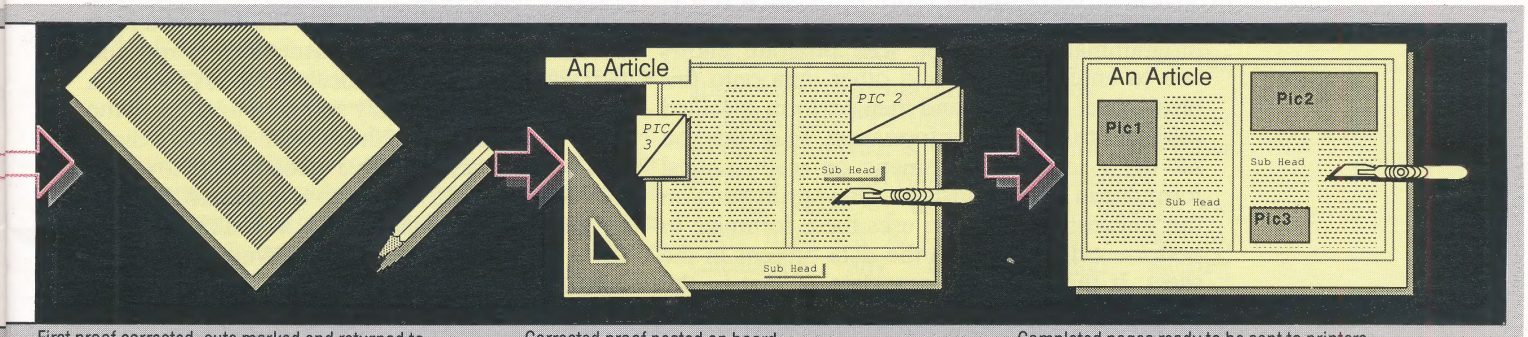
Taken one stage further, a rationalised system would ensure that the computerised typesetting machine was capable of reading the disk on which the original edited version was stored. If this were the case, the article could undergo the translation from the author's original draft to its typeset form existing only as a series of magnetic imprints on a disk until the typesetting was produced.

This is known as a 'direct entry' system — the text for the article is only keyed in once by its author and is subsequently installed in other machines by loading from disk or by electronic transmission between sections of a larger, homogeneous, computerised system.

The capabilities of the new technology extend much further than text manipulation, however. Recent developments in computer operating environments, particularly WIMP (Windows, Icons, Mouse Program) systems, have opened the way for the computer to impinge on the work of the designer. These new systems allow the designer to layout a page on screen, often including digitised photographs and diagrams. Different typefaces and arrangements of text can be simply manipulated using a controlling mouse. A final version of the page can then be printed directly using a high-quality printer.

Although the production of newspapers differs in some respects from the production of magazines, many of the same principles apply. Indeed, the rapid processing requirements of daily newspapers, together with their high sales volume, means that the new technology is likely to have a greater impact in this area of publishing. The technology currently used in Fleet Street was largely built in the 1930s and uses techniques developed in the late Victorian era. However, there are signs of impending changes within the newspaper industry.

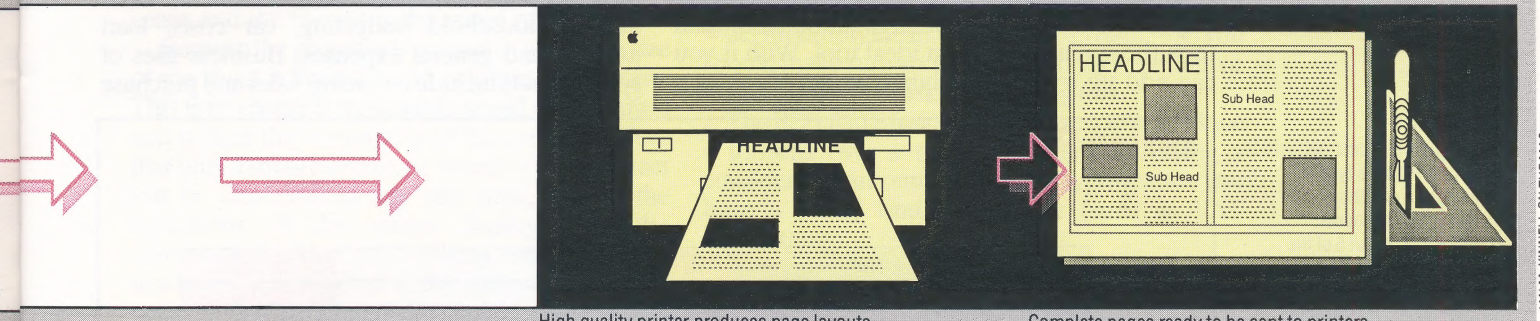
Already many provincial newspapers, in response to the challenge of cheaply produced free advertising sheets, have successfully applied computerised production methods. There are plans for small, more flexible companies to challenge the Fleet Street giants' monopoly of the national newspaper market by producing a daily with about one fifth of the staff normally required by a national paper using direct entry, on-screen layout and computerised printing methods.



First proof corrected, cuts marked and returned to typesetters

Corrected proof pasted on board

Completed pages ready to be sent to printers



High quality printer produces page layouts

Complete pages ready to be sent to printers

IAN McKINNELL ON THE MAC

One interesting spinoff of producing a newspaper electronically is in its distribution around the country. Traditionally, national newspapers print in one, or possibly two, centres (normally London and Manchester) and then distribute the printed copies by road and rail around the country. With an electronically produced newspaper, it is suggested that the method of distribution can be radically altered. Although the paper may be produced in one place, it can be printed at five or six local provincial presses by transmitting the paper electronically over ordinary telephone links, reducing distribution costs significantly. Furthermore, it becomes easy to produce local editions of the national paper, additional pages can easily be added to the main content of the paper at the local centres.

In the second part of our look at computers in publishing, we'll see how the new electronic publishing techniques are finding their way into the microcomputer market, and show how a magazine can be produced, from author to finished product, on a single micro.

A CLEAN SHEET

We begin a project to design our own spreadsheet program in BASIC for the four most popular home micros. After discussing the structure and applications of spreadsheets in general, we outline several of the main facilities that our program will feature. The Commodore 64 graphics routines are also given.

A spreadsheet program can best be thought of as an electronic sheet of paper, with a built-in programmable calculator. The program allows you to enter numbers and other data onto this blank sheet of paper, and then perform specified calculations using this information. To do this you set up a series of arithmetic or mathematical formulae in the spreadsheet. On your instruction the spreadsheet program uses these formulae to perform calculations on the data present in the sheet. It then puts the new data — that is, the results of the calculations — into the sheet at the positions specified by the formulae.

Thus, if you have a problem or application that requires you to work with large amounts of data then the spreadsheet is an ideal tool. With it you can add up rows and columns at the touch of a button — a feature that is useful for sales totals, applying mathematical formulae to a series of data, calculating percentages for VAT and income tax, and so on. In short, you can do anything with a spreadsheet that you can do with pencil, paper and calculator, only you can do it much faster and more accurately. Another useful feature of good spreadsheets is that once you have set up one to perform a particular set of functions, you can use that sheet again and again with different data each time. This is an extremely helpful facility if you have a regular set of calculations to do, such as weekly travelling expenses or sales commissions.

For example, if you run a small business, you can claim back the VAT you pay out on purchases against the VAT that you have to pay to the taxman. Obviously, before you can do this you have to work out the total VAT you have charged on the items you have sold and subtract the total VAT you have already paid on the items you have purchased. Using the normal method of paper and pen you would have to calculate the VAT on each individual item manually and then total up all the VAT at the end, a method which is both tedious and prone to error. With a spreadsheet, however, all you have to do is set it up to calculate the VAT by entering formulae — say, to multiply all the items in one column by 15 and divide by 100. The cost of all the purchases can then be entered and the spreadsheet does the rest.

The types of applications that spreadsheets can be used for are numerous. Around the home these include household budgeting, car costs, loan analysis and general expenses. Business uses of spreadsheets include invoicing, sales and purchase

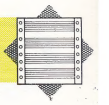
Balancing Act
The example spreadsheet layout given here shows a typical bookkeeping problem — balancing VAT accounts. Columns 1, 2, 4 and 5 of the spreadsheet must be set up by the user and the spreadsheet programmed to calculate the figures in columns 3 and 6 and the VAT totals. For example, figures in column 3 can be found by multiplying the corresponding figure in column 2 by 15/115, and programmed by setting up formulae in the column 3 cell as shown. It is often the case, as here, that similar formulae are required for an entire row or column of the spreadsheet. So that the user does not have to set these up manually, most spreadsheets include a replicate function that allows the formula in one cell to be repeated across a row or down a column

Taxing Problem

			A2*15/115			A5*15/100
			B2*15/115			B5*15/100
			C2*15/115			C5*15/100
			D2*15/115			D5*15/100

	1.BOUGHT	2.GROSS COST	3.VAT	4.SOLD	5.PRICE	6.VAT
A	MICRO	399.99	52.17	S/SHEET ART.1	120.00	18.00
B	PRINTER	249.99	32.60	CONVERT PROG	2000.00	300.00
C	PAPER	20.00	2.60	S/SHEET ART.2	120.00	18.00
D	INK RIBBON	7.24	0.94	WIDGET DATA	50.00	7.50
E	PENCILS, ETC	2.42	0.31			
F	DESK	92.47	12.06			
G	DISK DRIVE	199.95	26.08			
H	LAMP	24.87	3.24			
I	MIDNIGHT OIL	57.32	7.47			
J	BLOOD	10.57	1.37			
K	SWEAT	1.00	0.13			
L	TEARS	0.50	0.06			
M		VAT PAID	139.03		VAT CHARGED	343.50
N					NET VAT	204.47

CAROLINE CLAYTON



ledger, budgeting, forecasting and planning, cost analysis, profit and loss, job costing and estimating, sales commission calculations, project modelling — to name but a few. The benefits to the user are many and include increased productivity, accuracy and improved presentation.

The spreadsheet program which we will be designing in this series will give the BASIC programmer an insight into how commercial spreadsheet programs are written and how they work. As each part of the program is printed, a detailed explanation of how that section works and what it does will be given. The program has been written for the Spectrum, Commodore 64, BBC Micro and Amstrad CPC 464/664 computers; for each section of the program a Spectrum and a Microsoft-type BASIC listing will be printed, along with details for implementing the program on the other machines. The program is intended to be illustrative; it could be used as the basis of a much more sophisticated program — depending on your level of skill as a programmer and the amount of time you wish to devote to it.

Due to the complex nature of spreadsheets, we have concentrated on reproducing the main features of a spreadsheet program, with the result that there is only minimal error checking within the program. As a consequence, you should take great care when entering your data and using the program, as any errors will generally cause the program to crash. There are two reasons for the lack of error checking: the first is the constraint of space and the second is the speed of operation.

Because the program is written in BASIC, the size of the sheet is limited to 15 columns by 15 rows. This is to ensure a reasonable speed of operation and reduce the complexity of the program. Note that only a small portion, or 'window', of the sheet can be displayed at any one time, due to the limitations of the screen displays. On the Commodore 64, BBC Micro and Amstrad machines, this window is five columns by seven rows; on the Spectrum it's four by six.

CELLULAR STRUCTURE

Each of the 225 boxes on our spreadsheet is called a *cell*. These cells are labelled (on the first row) A1, A2 and so on to A15; B1, B2 and so on (on the second row) right down to the fifteenth row (O1, O2 and so on to O15). To move the cursor around the sheet, you simply use the cursor keys on your machine. The current cell that the cursor is pointing to is displayed at the top left-hand corner of the screen and the cell itself is illuminated.

In each cell you can do one of two things: either enter numeric data into the cell or set up a new formula for dealing with data. The data is displayed on the screen in the appropriate cell, and you can have only five digits per cell. If there is a formula present at the current cell it is displayed on the input line at the bottom of the screen. To enter a new formula simply select the enter formula option and type in your new formula. The formula evaluator in this program is quite a sophisticated one, evaluating the formula from left to right and with normal arithmetical precedence. The operators it can handle are +, −, *, /, ^ and parentheses. The operands allowed are cell names and real or integer numbers. Thus if you wished to add up the data in cells A1, A2 and A3, and then put the answer in cell A4, you would enter the formula A1+A2+A3 at cell A4. Again, should you wish to calculate the VAT on the value at say B1, then you would put the formula B1*0.15 at cell B2 and the formula B1+B2 at cell B3.

SPREADSHEET FUNCTIONS

Our BASIC spreadsheet program includes many features that make it a powerful tool for home use. These include:

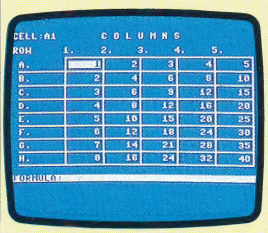
- A HELP screen, which prints a list of the options available to the user.
- A TEMPORARY STORE function that allows you to save the sheet currently displayed. This is particularly useful when you wish to use the spreadsheet to try out different values.
- You can also RESTORE a temporarily saved sheet to the screen (that is, once you've tried out other values, you can get your original data back).
- The CALCULATE function works through the sheet from left to right and top to bottom, calculating any formulae and altering corresponding cell contents as it proceeds. Therefore, you should make sure that you have arranged your data in an order that will not affect these calculations.
- CLEAR simply clears the sheet of all data — so you should make sure you've either stored your data or saved it to tape or disk before using this.
- The REPLICATE function allows you to reproduce a formula over a range of cells, thus saving a considerable amount of time and typing when you have a lot of similar formulae (such as VAT calculations).
- You can LOAD and SAVE the data in your sheet from or to tape.
- You can also LOAD and SAVE formulae independently of the data in the sheet. This is useful as it allows you to use a sheet that has been set up for frequent use (such as calculating weekly travelling expenses) with new data and without the need to set up the sheet manually each time.
- The TAB function allows you to move the cursor to any position on the sheet, without having to work your way across the screen with it.

We give here the listings for setting up the screen and handling the graphics on the Commodore 64. In the next instalment, we will give the corresponding listings for the BBC Micro, the Spectrum and the Amstrad machines. This is the only section of the spreadsheet program that will be given as separate listings. This is because the screen handling for each machine is totally different, whereas the rest of the program is similar for each computer (except the Spectrum version, which employs different string-handling methods).

Graphics Routines

Although much of the spreadsheet program can be given in the form of a common listing, the methods by which the four micros handle the display of information on the screen are all different. To produce attractive displays and screen-handling on each machine, we will therefore provide separate graphics routines listings for each computer. We start here by providing the graphics-handling listings for the Commodore 64.

These listings handle the printing of the spreadsheet grid, printing data in the spreadsheet cells and the movement of the spreadsheet cursor. After typing in the appropriate section of program and running it, you should see the grid on the screen (together with an initial set of data) and be able to move the cursor around the sheet. However, most of the spreadsheet functions outlined in this instalment will not yet be available, as they are the subject of future parts of the project



LIZ HEANEY

Commodore 64

```
10 REM **** CBM 64 SPREADSHEET ****
15 REM ** SET UP CBM CHARACTER STRINGS *
*
20 FOR I=1 TO 7:L7$=L7$+CHR$(195):NEXT
30 L5$=LEFT$(L7$,5):T$=CHR$(178):B$=CHR$
(194)
40 S5$="     ":X$=CHR$(219):I$=CHR$(177)

100 GOSUB 3000:REM SETUP ARRAYS & VARIAB
LES
110 GOSUB 1000:REM PRINT SCREEN
120 GOSUB 1700:REM PRINT DATA ON SCREEN
130 GOSUB 1100:REM MAIN KEYBOARD SCANNIN
G ROUTINE
999 STOP
1000 PRINT CHR$(147);CHR$(145);CHR$(5):P
OKE 53280,6:POKE 53281,6
1005 PRINT "                C O L U M N S"

1006 PRINT
1007 PRINT "ROW     1.    2.    3.    4.
    5.    "
1010 PRINT "        "CHR$(176);L5$;T$;L5
$;T$;L5$;T$;L5$;T$;L5$;CHR$(174)
1020 FOR C=1 TO 7
1030 PRINT " "CHR$(C+64);".     "B$;S5$;
B$;S5$;B$;S5$;B$;S5$;B$;S5$;B$
1040 PRINT " ";L7$;X$;L5$;X$;L5$;X$;L5$;
X$;L5$;X$;L5$;CHR$(179)
1050 NEXT C
1060 PRINT " "CHR$(C+64);".     ";B$;S5$
;B$;S5$;B$;S5$;B$;S5$;B$;S5$;B$
1070 PRINT " ";L7$;I$;L5$;I$;L5$;I$;L5$;
I$;L5$;I$;L5$;CHR$(189)
1080 RETURN
1100 P$=CHR$(Y+64)+MID$(STR$(X),2,2):PRI
NT CHR$(19);"CELL:";P$;" "
1110 GET A$:IF A$="" THEN 1110
1120 IF A$=CHR$(29) THEN 1200:REM MOVE R
IGHT
1130 IF A$=CHR$(157) THEN 1300:REM MOVE
LEFT
1140 IF A$=CHR$(17) THEN 1400:REM MOVE D
OWN
1150 IF A$=CHR$(145) THEN 1500:REM MOVE
UP
1152 IF A$=CHR$(133) THEN GOSUB 6000:REM
 F1 PRINT HELP SCREEN
1155 IF A$=CHR$(137) THEN GOSUB 2000:REM
 F2 INPUT FORMULA
1158 IF A$=CHR$(134) THEN GOSUB 5150:REM
 F3 STORE CURRENT SHEET
1160 IF A$=CHR$(135) THEN GOSUB 2300:REM
 F5 CALCULATE SHEET
1165 IF A$=CHR$(13) THEN RETURN
1170 IF A$)"0" AND A$<"9" THEN GOSUB 210
0:REM ENTER NUMERIC DATA
1180 IF A$=CHR$(139) THEN GOSUB 5000:REM
 F6 CLEAR SHEET
1185 IF A$=CHR$(138) THEN GOSUB 5100:REM
 F4 GET PREVIOUS SHEET
1187 IF A$="G" THEN GOSUB 5200
1188 IF A$=CHR$(136) THEN GOSUB 5700:REM
 F7 DECIDE COL OR ROW
1189 IF A$=CHR$(140) THEN GOSUB 7000:REM
 F8 LOAD/SAVE ROUTINES
1190 GOTO 1110:REM BACK TO START
1200 REM ****** MOVE RIGHT ******
1210 IF X=15 THEN 1100
1220 IF X=H2 THEN GOSUB 1600:X=X+1:H1=H1
+1:H2=H2+1:GOTO 1270
1230 GOSUB 1600:LET X=X+1:GOSUB 1650:GOT
O 1100
1270 GOSUB 1800:GOSUB 1700:GOTO 1100
1300 REM ****** MOVE LEFT ******
1310 IF X=1 THEN 1100
1320 IF X=H1 THEN GOSUB 1600:X=X-1:H1=H1
-1:H2=H2-1:GOTO 1370
1330 GOSUB 1600:LET X=X-1:GOSUB 1650:GOT
O 1100
1370 GOSUB 1800:GOSUB 1700:GOTO 1100
1400 REM ****** MOVE DOWN ******
1410 IF Y=15 THEN 1100
1420 IF Y=V2 THEN GOSUB 1600:Y=Y+1:V1=V1
+1:V2=V2+1:GOTO 1470
1430 GOSUB 1600:LET Y=Y+1:GOSUB 1650:GOT
O 1100
1470 GOSUB 1850:GOSUB 1700:GOTO 1100
1500 REM ****** MOVE UP ******
1510 IF Y=1 THEN 1100
1520 IF Y=V1 THEN GOSUB 1600:Y=Y-1:V1=V1
-1:V2=V2-1:GOTO 1570
1530 GOSUB 1600:LET Y=Y-1:GOSUB 1650:GOT
O 1100
1570 GOSUB 1850:GOSUB 1700:GOTO 1100
1600 REM ***** TURN CURSOR OFF *****
1610 CU=1023+40*(V(Y+1-V1)-1)+H(X+1-H1):
POKE CU,PEEK(CU)-128
1620 POKE CU+1,PEEK(CU+1)-128:POKE CU+2,
PEEK(CU+2)-128
1630 POKE CU+3,PEEK(CU+3)-128:POKE CU+4,
PEEK(CU+4)-128:RETURN
1650 REM ***** TURN CURSOR ON ******
1660 CU=1023+40*(V(Y+1-V1)-1)+H(X+1-H1):
POKE CU,PEEK(CU)+128
1670 POKE CU+1,PEEK(CU+1)+128:POKE CU+2,
PEEK(CU+2)+128
1680 POKE CU+3,PEEK(CU+3)+128:POKE CU+4,
PEEK(CU+4)+128
1690 GOSUB 1900:RETURN
1700 REM **** PRINT DATA IN SHEET ****
1710 FOR I=0 TO 7
1720 PRINT CHR$(19);:FOR C=1 TO V(I+1)-1
:PRINT CHR$(17);:NEXT C
1730 FOR J=0 TO 4
1735 P$=MID$(STR$(MAT(I+V1,J+H1)),2)
1740 PRINT TAB(H(J+1)-1);"     ";CHR$(14
5)
1745 PRINT TAB(H(J+1)+4-LEN(P$));P$;
1750 NEXT J,I
1760 GOSUB 1650:RETURN
1800 REM **** PRINT COLUMN NUMBER ***
1810 PRINT CHR$(19);CHR$(17);CHR$(17);CH
R$(17);
1820 FOR I=H1 TO H2:PRINT TAB(7+6*(I-H1)
)I;CHR$(157);". ";
1830 NEXT I:RETURN
1850 REM **** PRINT ROW NUMBERS *****
1860 PRINT CHR$(19);:FOR I=1 TO 5:PRINT
CHR$(17);:NEXT
1870 FOR C=V1 TO V2
1880 PRINT TAB(1)CHR$(C+64);CHR$(17)
1890 NEXT C:RETURN
1900 REM *** FORMULA OF CURRENT CELL **
1920 LET D$=F$((Y-1)*15+X)
1930 GOSUB 1950:REM MOVE CURSOR TO INPUT
 LINE
1935 PRINT CHR$(18);"FORMULA:
                "
1940 PRINT CHR$(145);CHR$(18)"FORMULA:";
D$
1945 PRINT CHR$(19):RETURN
1950 REM ***** CURSOR TO INPUT LINE ***
1960 PRINT CHR$(19);:FOR K=1 TO 22:PRINT
 CHR$(17);:NEXT K
1970 RETURN


3000 REM ******   SETUP ARRAYS *******
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20)
,G$(20)
3020 FOR C=0 TO 4
3030 H(C+1)=6*C+10:REM CALC X POS
3040 NEXT C
3050 FOR C=1 TO 8
3060 V(C)=2*C+4:REM CALC Y-POS
3070 NEXT C
3075 X=1:Y=1
3080 H1=X:H2=X+4:V1=Y:V2=Y+7
3090 REM *******  SHEET ARRAYS ********
3100 DIM MAT(15,15):DIM MC(15,15)
3110 FOR I=1 TO 15:FOR J=1 TO 15
3120 MAT(I,J)=I*J
3130 NEXT J,I
3140 DIM F$(225)
3150 RETURN
```

PLAY ON WORDS

In this final article in our series about FORTH, we turn our attention to the way in which the language manipulates strings. To illustrate the processes involved, we include some word definitions that simulate some of BASIC's string handling facilities.

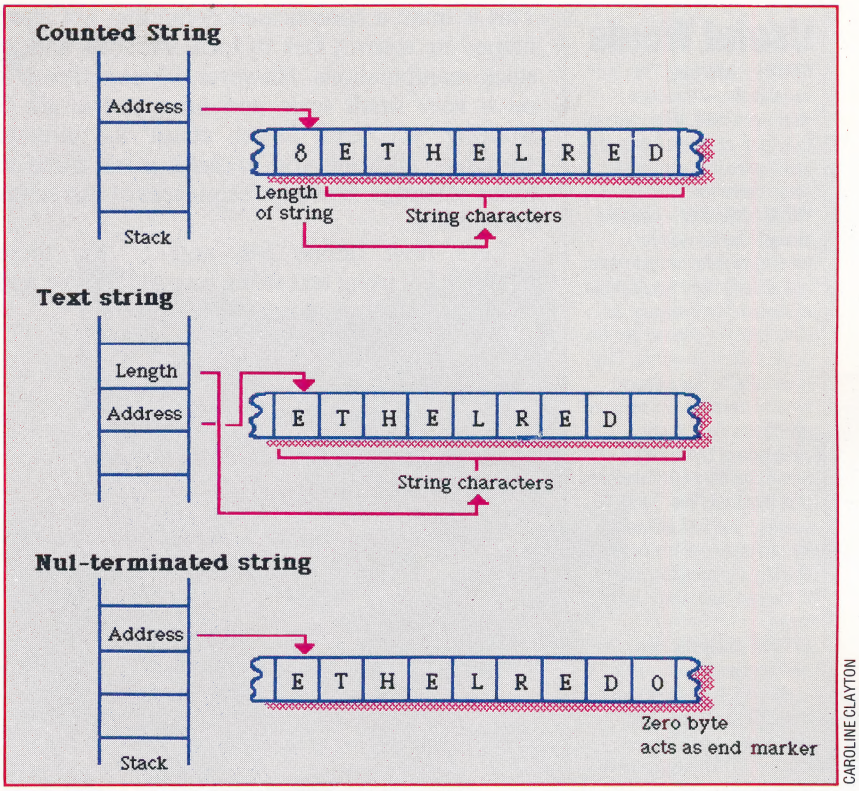
You have already seen the ." word, which is quite easy to use, but other string manipulations are more complicated. There are two main problems: in what area of memory to store the string, and how to store it. Such considerations are rather tricky, so naturally you will want to define powerful words that let you forget the details when you use them.

First, there are two main places to store a string — in the ALLOTted parameter fields of words in the dictionary, and in an area called the 'pad'. The pad is for very temporary string storage, because it is often used by the FORTH system. For instance, if anything is enclosed in the dictionary, or if a FORTH word is interpreted from the keyboard, the pad may move or be overwritten. Secondly, there are two ways in FORTH of storing a string. In both cases the ASCII characters of the string are stored in order somewhere in memory; the differences lie in how you specify the length of the string.

In a *counted string*, the length (less than 255 bytes) is stored as a single byte immediately before the characters. To refer to such a string you can use a single number, which is the address of this count byte. Thus, although you can't put the string itself on the stack, once you've set the string up you can put its address on the stack. For a *text string*, the length is not stored as part of the string at all. This means that to refer to the string you need two numbers, the address of the first character as well as the length.

Counted strings are easier to handle on the stack (the additional number needed for text strings makes stack manipulations disproportionately complicated). However, they are limited to 255 bytes and they can't be used for substrings — strings that form parts of larger strings — because you'd have to insert a count byte for the substring in the middle of the whole string.

Let's look at some words that define string variables. A string variable keeps its text in its parameter field, so once you've ALLOTted a certain amount of space for a string, it remains fixed, so that you'll never be able to assign longer strings to it. $VARIABLE therefore has you specify not only an initial value, but also a maximum length for subsequent assignments. The maximum length may need to be larger for the initial value.



CAROLINE CLAYTON

A useful word is ASCII. Most FORTHs have this built in, but it is not required by the standard. It stacks the ASCII code of the character following it, and if it's used in a colon definition, it compiles a numeric literal for this number so that it gets stacked at run time.

```
32 CONSTANT BL ( ASCII code for a space )

: ASCII ( -- ASCII code ) ( used in form ASCII
        character )
    BL WORD ( get next word )
    1+ C@ ( and get the ASCII code for its first
          character )
    STATE @ IF ( if compiling colon definition )
       [COMPILE] LITERAL ( LITERAL is immediate )
    THEN
; IMMEDIATE ( ASCII has special compiling actions )

: $ VARIABLE ( size -- ) ( used in the form — )
        ( size $VARIABLE name initialiser" )
        ( the parameter field is set up to contain 1 byte
        for the )
        ( maximum length and then a counted string
        for its value )
    CREATE
    255 MIN ( ensures size not too big )
    1 ALLOT ( for max. length )
    ASCII "WORD ( size, initialiser as counted string)
    DUP C@ ROT MAX ( initialiser, max. length )
    DUP HERE 1— C! ( fill in max. length )
    HERE SWAP 1+ ALLOT ( initialiser, address to put
    it )
    OVER C@ 1+ CMOVE ( copy in initialiser )
DOES> ( pfa -- pfa+1 ) ( stacks value as counted
          string )
    1+
;
```

String Formats

FORTH allows strings to be held in memory in three different ways. Each method uses an address number (which can be simply manipulated using the FORTH stack) that points to the area of memory containing the string. However, the three techniques use different methods of determining the length of the string. The 'counted string' format stores this information in the first byte of the memory area containing the string, while the 'text string' format holds this number separately on the stack — allowing stack manipulation of it. Some versions of FORTH support the 'nul-terminated' format, in which the end of the string is simply marked by a zero byte

Useful Words

STATE (-- address) This is a variable that immediate words can read. If its value is true (non-zero) then they are being used in compiling a colon definition.

WORD (delimiter -- counted string) This is used for reading the next word in what has already been typed at the keyboard. For instance, it's what CREATE uses to get the name of its new words. This is usually 32 (for a space), but it is possible to use other delimiters like " in our definition of $VARIABLE. Some standard FORTH words play this trick, like ." and (using " and) as delimiters. Be careful to use the word quickly. It is probably stored in the pad, and won't last long there.

EXPECT (address, max. number characters --) This is used for getting fresh input from the keyboard. The characters, up to the maximum number specified, are read into memory starting at the given address. It stops reading when you press ENTER or you have typed the maximum number.

SPAN (-- address) This variable is set by EXPECT to show how many characters were read in. Only FORTH-83 has SPAN; older FORTHs put NUL characters after the string in memory to show where it finishes.

PAD (-- address of the pad)

CMOVE (from address, to address, number of characters --) This copies the specified number of bytes, starting at the 'from' address, to the memory starting at the 'to' address, so it is used for copying strings. There is a trap to be wary of, however. If the 'to' area of memory is after the 'from' area, and the two areas overlap, then some of the bytes in the 'from' area will be overwritten before they have been read, so it won't be a faithful copy.

CMOVE> (from address, to address, number of character --) This word, in FORTH-83 only, avoids the trap in CMOVE by copying bytes in a different order. However, it has a trap of its own when the 'to' area is before the 'from' area.

COUNT (counted string -- address, length of text string) The string remains the same, but COUNT converts between the two different ways of referring to it on the stack

This is quite a complicated definition, not at all helped by FORTH's lack of local variables and its stack manipulations. However, it's complexity is on a very small scale and once you've got it working you can forget about its internal workings, allowing you to think much more in terms of strings, instead of sequences of characters in memory.

Let's now define $@ and $! for these $VARIABLES, using text string format (address and length) on the stack to refer to the values.

```
: $@ ( pfa+1 for string variable -- text string )
   COUNT
;

: $! ( text string, pfa+1 for string variable -- )
     ( truncates the value if it's too long for the
     variable )
   DUP 1- C@ ROT      ( addr, pfa, max. length,
                        actual length )
   MIN                ( addr, pfa+1, length to use )
   OVER OVER SWAP     ( addr, pfa+1, length, length,
                        pfa+1 )
   C!                 ( store new length )
   SWAP 1+ SWAP       ( addr, pfa+2, length )
   CMOVE
;
```

You can now set up string variables, as in:

```
0 $VARIABLE DIGITS 0123456789"
```

(The 0 is automatically increased to 10 to cope with the 10 characters actually present. But after that, the maximum length of the string variable is fixed at 10.)

```
DIGITS $@ TYPE
```

(displays 0123456789)

```
255 $VARIABLE XPAD XXX"
```

(an alternative, more stable pad with space for 255 characters.)

It would now be nice to be able to change the values of strings from the keyboard. Here are two words, " and $INPUT, that let you do the equivalent of the BASIC assignment statements LET A$ = "..." and INPUT A$.

```
: "       ( -- address, length of text string )
          ( used outside colon definitions in form -
          "text" )
   ASCII " WORD COUNT ( get text string )
   XPAD $!            ( put it in XPAD )
   XPAD $@            ( leave address and length )
;
```

The word " gets its string in the same way as words like VARIABLE get the new name, which is why it won't work properly in colon definitions. You can, however, use it as in:

```
20 $VARIABLE PET cat"
"parrot" PET $! PET $@ TYPE
"pangolin" PET $! PET $@ TYPE
```

Here is a word to imitate input statements:

```
: $INPUT  ( pfa+1 of string variable -- )
          ( waits for you to type a string and copies
          it to the variable. )
   DUP DUP 1+      ( pfa+1, pfa+1, pfa+2 )
   SWAP 1- C@ 2-   ( pfa+1, pfa+2, max. length - 2 )
   EXPECT
   SPAN @ SWAP C!
;
```

The program informs EXPECT not to get more characters than there is room for in the variable. However, one version of FORTH allows an extra NUL character to be written after the string. So to be on the safe side, 2 has been subtracted from the maximum length. Otherwise, the extra NUL could corrupt the header for the next word in the dictionary.

In FORTH-79 or figFORTH, EXPECT will write one or two NULs in any case, which in fact are vital. The older FORTHs do not have SPAN and so you will have to write your own word to find the length of the string by counting characters up to the NUL.

Now at last we can define a FORTH version of that most trivially satisfying of BASIC programs:

```
10 PRINT "What is your name?"
20 INPUT A$
30 PRINT "Pleased to meet you, ";A$;"."
40 PRINT "Take care now! Bye."
```

A comparable program in FORTH, which uses commands that we define ourselves, is:

```
30 $VARIABLE NAME

:HI!
   ." Hi there! I'm the computer. Who are you?" CR
   NAME $INPUT
   ." Wow! That's an incredible name, erm," NAME
   $@ TYPE
   "." ( type full stop ) CR
   ." Look after yourself, you hear? Have a nice day."
   CR
;
```

BASIC's great strength is that it is very easy to write short programs, drawing on its useful facilities for things like floating point arithmetic, arrays and strings. Where BASIC loses its advantage is in writing larger programs. You can never escape the low-level structure of the program — line numbers, for instance. By contrast, look at our FORTH word HI!. You should be able to understand this without much trouble, given a rough understanding of what $INPUT, $@ and other words do. If you want to look at it more closely, you can go back to the definitions of these words, but even so you only need to know *what* they do -- what they expect on the stack and what they leave behind — and ignore *how* they do it. Thus in FORTH you don't need to keep the low level structure in mind.

The correct moral to draw is this: even if FORTH starts off less powerful than another language, you can use its extendability to redress the balance. What is more, you can continue to extend it so that it far surpasses the other language.

BIG BROTHER

CHRIS STEVENS

One would expect any new product from a company with the immense sales and stature of IBM to make an enormous impact on the market. The new PC/AT, with a possible three Mbytes of accessible memory and the highly advanced Intel 80286, sets a new standard for microcomputers.

Because of IBM's reputation, microcomputers in 1982 suddenly became fashionable in the business world, with the PC taking a dominant share of the market. This led to certain consequences, the full impact of which could not have been foreseen even by IBM itself. Much of the technology used in the original IBM PC was bought in from other suppliers, such as the Intel 8088 processor, the $5\frac{1}{4}$in disk drives and, most importantly, the operating system PC-DOS. None of these components represented advanced technology for the time, resulting in much criticism, but the machine nevertheless sold in large quantities, which naturally generated an enormous software base. Other manufacturers, feeling themselves being edged out of the market, followed IBM's example and brought in identical technology to produce 'IBM clones' — computers that would run IBM software. The result of this was that the IBM PC became the *de facto* industry standard.

Since then, the falling price of memory chips, coupled with ever more powerful processors, has resulted in business machines being equipped with 256 Kbytes as standard (the original PC had only 64 Kbytes of RAM), which in turn has led to software houses producing more complex and memory-hungry programs.

The current height of this trend is represented by the IBM PC/AT (Personal Computer/ Advanced Technology). This massive computer, beginning to approach the power and capacity of a minicomputer, is equipped with one of the latest processors: the Intel 80286. This chip is equipped with a 16-bit data bus and a 24-bit address bus similar to that on the 8086 chip used in both the Olivetti M-24 and the Apricot range, enabling the machine to address over 16 Mbytes of memory. However, the chip also has the facility for 'virtual memory management', which means that the actual memory capacity it can use is over a gigabyte (1,000 Mbytes).

Virtual memory management is a process whereby the processor can treat its available RAM and backup storage as 'main memory'. When running a program, all of the data and program may not fit into the available RAM, and so the rest will be held on high-speed disks. When the computer requires a piece of information and sees that it is not in RAM, the relevant data will be read off the disk into an unused area of memory, where it can be acted upon accordingly. Thus the computer can *appear* to have far more memory than it actually has.

This technique has been tailored for use on the PC/AT. The original purpose of virtual memory management was to use RAM in conjunction with cheaper methods of storage such as disks. With the drop in price of RAM chips, however, this is not a concern of the PC/AT system. The problem lies in the operating system.

PC-DOS, and the closely related MS-DOS, are capable of addressing 640 Kbytes of memory. When PC-DOS was developed, this was considered ample memory for any foreseeable application. However, many modern integrated software packages are rapidly approaching the limit of the OS's capabilities. Thus some method had to be found to add extra memory within the restrictions imposed by PC-DOS.

The answer has been to treat the remaining memory as 'silicon disks'. These are banks of memory that can be of any size (although the default size on the PC/AT is 64 Kbytes) up to that of the addressable memory, which are treated by the processor as information held on a disk drive. This means that the processor cannot address the memory directly, but can fetch the information in blocks, which can be moved to addressable memory and manipulated there. When the blocks are no longer required, the updated information or

Advanced Technology
The entry of IBM into the personal computer market marked a change in business attitudes to microcomputers. The company's new IBM PC/AT is currently one of the most powerful micros available for business use. Running up to three times as fast as the IBM PC, the AT is also capable of accessing up to three megabytes of memory

IBM PC/AT

PRICE
£3,304 inc. VAT for the Base Model, Enhanced Model £4,972 inc. VAT. Keyboard, interface cards, monitor and second disk drive optional

DIMENSIONS
540×420×160mm

CPU
Intel 80286 running at 6MHz

MEMORY
Base Model 256 Kbytes, Enhanced Model 512 Kbytes expandable to 3 Mbytes

SCREEN
Text display 80×25 characters, giving a maximum resolution of 640×200 pixels

INTERFACES
Eight expansion slots available for a wide range of interfaces

LANGUAGES AVAILABLE
Every major commonly used language

DOCUMENTATION
As usual with IBM, the documentation is thorough, although a little staid

STRENGTHS
The PC/AT is one of the most powerful commercially available microcomputers in the world, built by the world's biggest computer manufacturer. It is hard to see how it can fail

WEAKNESSES
Incompatibility with some existing PC software

program can be 'written' back to the silicon disk. Of course, although they are treated as disks, the fact that 'silicon disks' are in fact RAM chips means that access is much faster than from a conventional disk drive.

The PC/AT is available in two basic configurations. The entry price system, known as the Base Model 1, contains 512 Kbytes of RAM and a single disk drive, while the Enhanced Model has an additional 20 Mbyte hard disk unit attached and a serial/parallel adaptor.

DISK DRIVES

The disk drives fitted to the PC/AT, although still 5¼in floppy disks, are great improvements on those fitted to the original versions of the IBM PC, which were somewhat slow and noisy with a capacity of only 160 Kbytes — those fitted to the PC/AT have a capacity of 1.2 Mbytes. This is partly due to improved technology in the design of the drives and also due to improvements in the DOS. The PC-DOS bundled with the machine is version 3, which reads and writes 15 sectors per track instead of the PC's eight. However, to maintain compatibility with previous versions of the IBM PC, the new machine can detect PC-format disks and adjust itself accordingly — PCs cannot, however, read PC/AT disks.

Apart from being updated to take advantage of the higher capacity disk drives, PC-DOS 3 has had some additional commands fitted to enhance its own performance. ATTRIB designates a file as read-only and LABEL enables you to allocate a 'volume label' to a disk. SELECT and COUNTRY are similar commands, with the former enabling you to choose the keyboard layout and time/date format required, and the latter automatically switching the time/date and keyboard layout to that of a particular country. SHARE permits file sharing, FCBS specifies the number of file control blocks that can be open at any one time, and DEVICE enables you to install the size and number of the 'virtual disks' within the computer. Finally, LASTDRIVE lets you set the number of drives that can be used by the system.

Concerning the thorny problem of compatibility, it is perhaps surprising to discover that the PC/AT is not completely compatible with the original PC. This is partly because the disk drives used on the new machine, although enhanced, are unable to cope with some of the sophisticated copy-protection techniques that have been developed for PC software. Another difficulty is that some of the chips within the PC/AT have been altered from the original, and although they are supposed to be compatible, software that directly interrogates these chips may find that some of the addresses have changed. The most notable of these is the BIOS ROM, at the heart of IBM PC compatibility, to which certain improvements have been made. Unfortunately, this means that some changes have also been made to the entry addresses of certain routines.

This problem of incompatibility extends to the processor as well. Although the Intel 80286 is made by the same company that produced the PC's 8088 processor, some of the instructions are different — software for one machine using the incompatible instructions will not run on the other. Finally, the original IBM PC contained DIP switches that have been replaced within the PC/AT by CMOS RAM.

Because of this, many software products have had to be adjusted to fit the new format, mostly with regard to their protection codes. Perhaps the most notable of these is the Microsoft Flight simulator (FS1) — which is considered as one of *the* tests of IBM compatibility. Microsoft has therefore introduced an upgraded package, FS2.

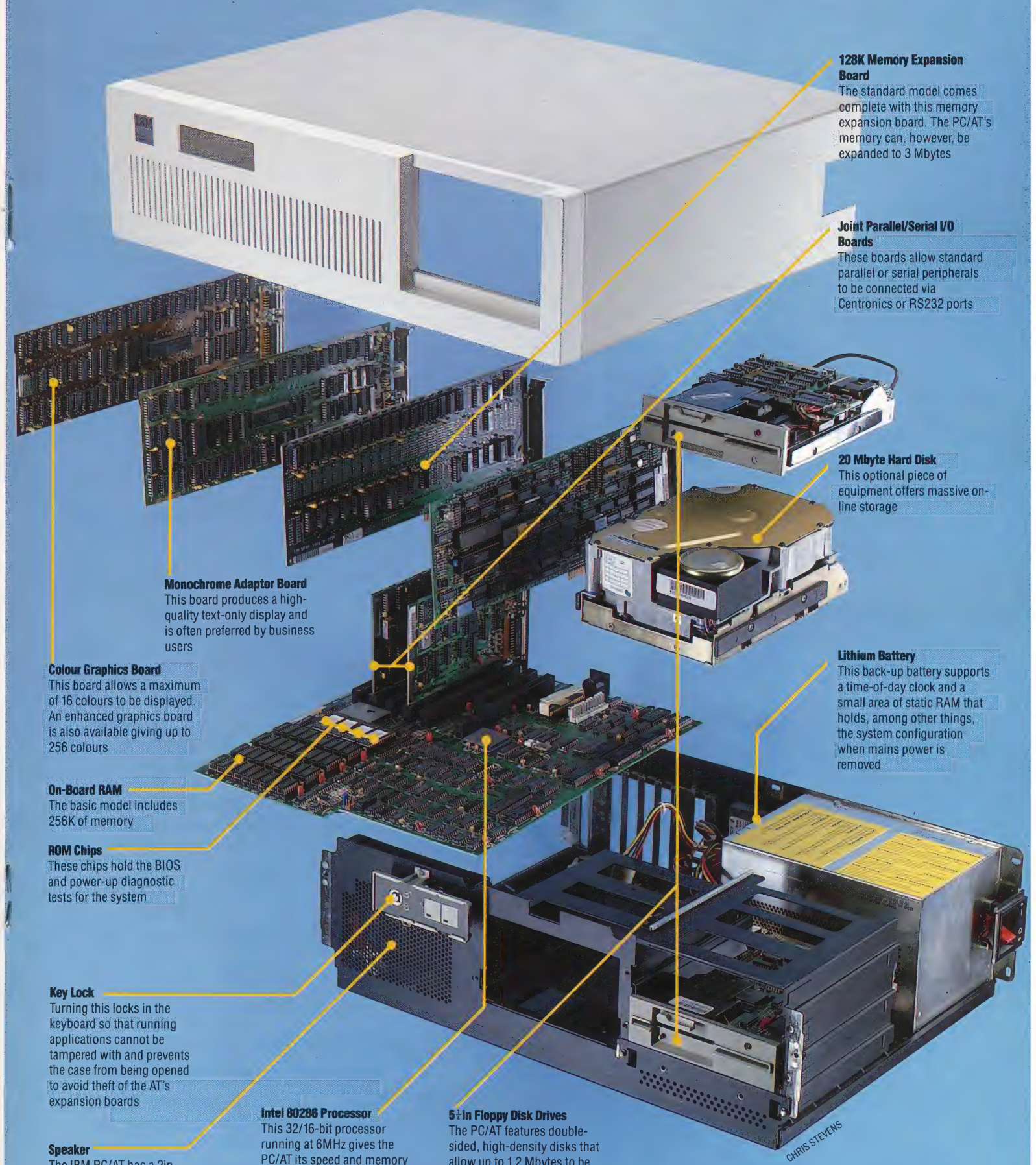
Despite these difficulties, the IBM PC/AT is an excellent machine to use, and almost all of the criticisms levelled at the original PC have been answered positively. Most notable among the improvements is the keyboard, which is undoubtedly the best, in terms of feel, that has ever been produced on a microcomputer.

As might be expected, the new machine is considerably faster than the original IBM PC — up to three times as fast in some applications. Coupling that with the vast memory this machine is capable of accessing, there is little doubt concerning the PC/AT's future.

Deceptive Appearances

The technique of virtual memory management was originally developed for use on mainframe computers, and is used to provide a computer with an appearance of having much more memory than it actually has. The principle behind virtual memory management is fairly simple. Often, when running a program or suite of programs together, there is insufficient memory available, and so the programs are kept on a fast-access backing store, such as a hard disk. When the programs are run, the computer only loads the parts of a program that are required to accomplish the task in hand. When the job is finished, the program will be erased and the next part of the program will be loaded. Provided that the process is fast enough, it will 'appear' that all the parts of the program are in memory and running simultaneously.

The practical application of virtual memory management that's most easily seen is in multi-tasking, where a computer will be running several entirely separate programs at once. For example, a computer might be required to access a load of data, format it and print it out, while at the same time maintain a network and word processing facilities. Here, while it may appear that all three programs are running at once, there will be periods when the programs are 'idling' — waiting for ready to receive, input or send messages from a peripheral. In this way, the programs can be kept on disk and only loaded and run when they are required. Meanwhile, the computer can utilise its entire memory to perform other tasks



128K Memory Expansion Board
The standard model comes complete with this memory expansion board. The PC/AT's memory can, however, be expanded to 3 Mbytes

Joint Parallel/Serial I/O Boards
These boards allow standard parallel or serial peripherals to be connected via Centronics or RS232 ports

20 Mbyte Hard Disk
This optional piece of equipment offers massive on-line storage

Monochrome Adaptor Board
This board produces a high-quality text-only display and is often preferred by business users

Colour Graphics Board
This board allows a maximum of 16 colours to be displayed. An enhanced graphics board is also available giving up to 256 colours

Lithium Battery
This back-up battery supports a time-of-day clock and a small area of static RAM that holds, among other things, the system configuration when mains power is removed

On-Board RAM
The basic model includes 256K of memory

ROM Chips
These chips hold the BIOS and power-up diagnostic tests for the system

Key Lock
Turning this locks in the keyboard so that running applications cannot be tampered with and prevents the case from being opened to avoid theft of the AT's expansion boards

Speaker
The IBM PC/AT has a 2in speaker built in

Intel 80286 Processor
This 32/16-bit processor running at 6MHz gives the PC/AT its speed and memory addressing capabilities

5¼in Floppy Disk Drives
The PC/AT features double-sided, high-density disks that allow up to 1.2 Mbytes to be stored on a single disk

CHRIS STEVENS

WHEN THE CHIPS ARE DOWN

The seven-segment LEDs and the power supply are the only remaining connections required for our DVM project. By referring to the main layout diagram on page 1552, we can complete the construction phase of the project and carry out a series of tests to ensure that all has gone well.

For reasons that were explained in the previous instalment five identical LEDs are used (see page 1552). It's not necessary to use sockets for the LEDs, but soldering directly to the rather small pins is rather tricky. If you do opt to use sockets (these are not listed in the parts list) remember that 24-pin, 28-pin and 40-pin DIL sockets have the pins correctly spaced (seven holes on a 0.1in matrix board).

The LEDs can be mounted on a small separate portion of matrix board and connected to the main board by a few inches of wire or ribbon cable. This will allow the display to be mounted in the window of a small instrument case if required. The fifth pins of LEDs 3, 4 and 5 have leads connected to them, which are for switching the decimal point.

A simple power supply is used, based on two

Parts List

No	Ref	Item	Maplin Number
5	LED1-5	Common anode 7-segment LED	FR39N
1	T1	12v transformer	WB25C
1	BR1	Bridge rectifier	QL38R
1	C1	680uF 40v electrolytic capacitor	FB79L
1	C2	0.33uF tantalum bead capacitor	WW57M
2	C3,C4	0.1uF tantalum bead capacitor	WW54J
2	RG1,RG2	3-terminal 5v regulator	QL28F
1	—	Heatsink	FL58N
2	D1,D2	1N4001 rectifier diode	QL73Q

Miscellaneous:

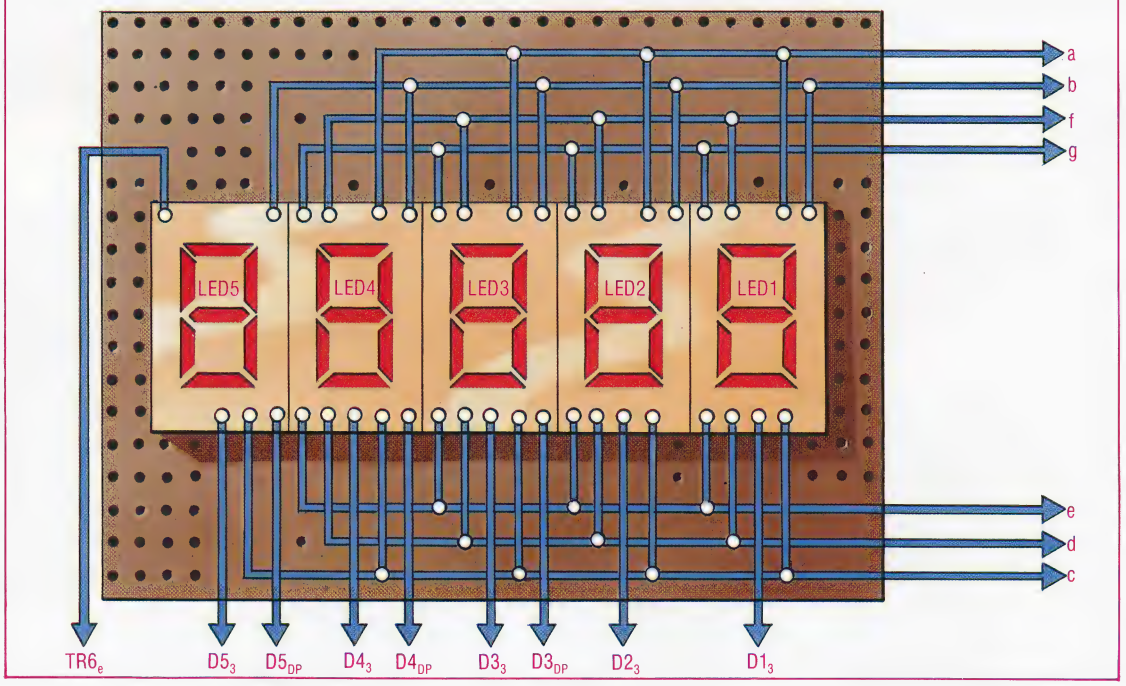
1.5m	Wire-wrap wire
0.5m	12-way ribbon cable

The LED Display

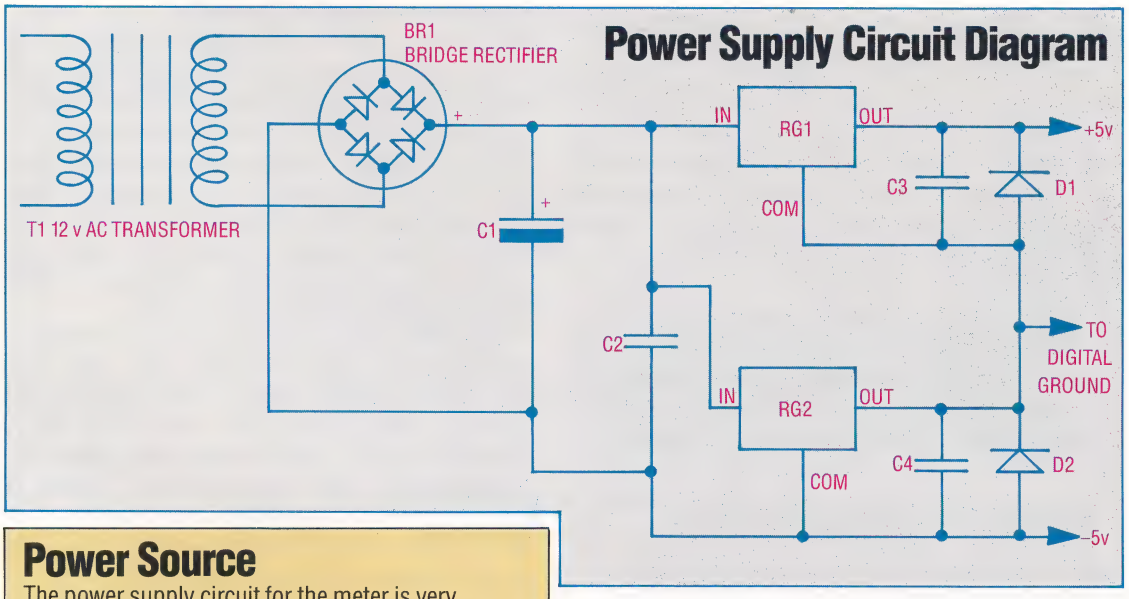
The five-digit LED display for our meter mounts on a separate piece of 0.1inch matrix board, as shown. The seven signals that light the LED bars (a to f) are common to the four rightmost digit displays, as these signals are multiplexed between all four digits. Use wire-wrap wire to make the necessary point-to-point connections. A short length of ribbon cable can then be used to connect these seven lines to the main board. Refer to the main board layout given on page 1553 to see where these lines should connect.

The digit enable signals from the main board connect to pin 3 of each LED display. Again, refer to the main board diagram. Reading this diagram from the right, connect $D1_3$ to D1, $D2_3$ to D2, $D3_3$ to D3, $D4_3$ to D4 and $D5_3$ to the rightmost connection of the three points marked D5. Note that the decimal point connections, marked Dn_{DP}, and $TR6_e$ should not be connected at this stage, as they are the subject of the final instalment

LED Display



KEVIN JONES



Power Source

The power supply circuit for the meter is very straightforward, using a bridge rectifier and two three-terminal voltage regulators to convert a 12v AC input current into +5v and –5v DC feeds for our main meter board. The capacitors in the circuit are there to reduce power 'spikes' — sudden fluctuations — in the power supplied by the transformer

'three-terminal' regulators. This configuration allows a single 12v AC supply and a single-bridge rectifier to be used. The two diodes need not be the type specified — any small rectifier diode will do the job. They prevent a condition known as 'latch-up', which might otherwise occur when two regulators are used in a bipolar power supply.

C1 can be any reasonably large electrolytic capacitor, but be sure that it has a sufficiently large working voltage (at least 35v). C2, C3 and C4 should be solid tantalum bead capacitors, which are very intolerant of reverse voltages. Be sure to connect them the right way round — '+' on the body indicates the positive side.

The power supply components (apart from the mains transformer) can be conveniently mounted on a tag board, such as Maplin's 36-tag board. RG2 only has to supply a few milliamps and should not need a heatsink, but RG1 is working much closer to its limits and should therefore be bolted to a small heatsink such as Maplin's FL58N. Note that the body of the three-terminal regulator is connected to the common (centre) lead. Although most heatsinks are made of anodised aluminium and are not very conductive (electrically), it's a sensible precaution to ensure that the heatsink doesn't touch any exposed wires or chassis metalwork.

POWER SUPPLY TESTS

Having soldered the power supply, connect the mains transformer to the mains (pay due attention to safety when you do this) and connect a 100ohm resistor between the +5v output and ground. If you have any kind of voltmeter (even a cheap analogue one) measure the voltage across the resistor. It should be within a few percent of 5v. If you have an oscilloscope, switch the input to DC at a sensitivity of about 10v. Put the probes across the 100ohm dummy load and check on the screen that there is no appreciable ripple. If there is, it can ruin the performance of the DVM.

Three-terminal regulators are wonderfully easy devices to work with, but they can generate high-frequency switching noise. The layout shown in the illustration (as opposed to the connections shown in the circuit diagram) will help to minimise this noise.

Once the power supply is tested and proven correct, connect it to the main DVM board, but *do not* insert the ICs at this stage. Use your multimeter to check for +5v at pin 11 of IC1, pins 4 and 8 of IC2 and pin 16 of IC3. Check also for +5v at the collectors of TR1 to TR5, and for −5v at pin 1 of IC1. Now switch to the ohms range and connect one probe of the meter to either the analogue or digital ground points. Touch the other probe to pin 3 of IC1, pin 1 of IC2 and pin 8 of IC3. The resistances shown on the meter should be very low (less than 1ohm).

IC3 is an ordinary TTL chip and can be handled without any special precautions — just plug it into the socket. Make sure, however, that the notch on the chip is at the top, as shown in the last instalment. IC2 and IC1 cannot be treated similarly. IC2 is the CMOS version of the NE555 timer chip, and it is said to be fully protected against electrostatic discharge, but it's better not to take any chances. IC1 will certainly be destroyed by incorrect handling, and if you haven't dealt with electrostatically sensitive devices before, here's how you do it.

First, tear off a couple of feet of aluminium kitchen foil and lay it on the bench. Then disconnect the power supply and place the circuit board in the centre of the sheet. Reach under the foil and press it firmly against the pin side of the circuit. IC1 and IC2 will be supplied in some kind of conductive packaging. This may take the form of black plastic foam, a black 'rubber' holder or a specially treated plastic bag. Whatever these two

chips come in, place them, complete with packaging, onto the aluminium foil.

Press your bare elbow onto the foil, and keep it there while removing the chip from its packaging. Keeping your elbow in position, insert the chip into its socket. When the chips are firmly inserted, you can take your elbow away and discard the foil. (It's all too easy to forget to remove the foil when you connect the power supply, so beware.)

Warning!

Any project that involves mains supply can be dangerous. Ensure that you switch off the mains supply before working on any part of the board

Voltage Regulator Layout

The components for the power supply rectification and regulation circuit should be mounted on a 36-tag length of tag board. This type of board is ideal for laying out circuits of this type. Simply solder the legs of the various components to the tags, as shown, and use covered leads to link the tags together. The AC input from the transformer voltage feed wires should be connected to the main multimeter board. The -5v feed connects to a point near the bottom of this board and the ground line connects to the main analogue ground point on the board. As the +5v feed has to supply several points on the main board, we recommend that you make this lead run around the edge of the main board and tap off where appropriate.

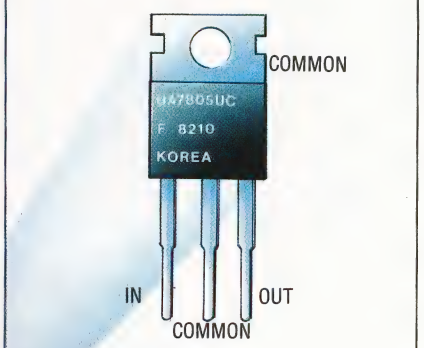
It is important to note the orientation of capacitors and diodes, bridge rectifier and regulators. In particular, the three-terminal voltage regulators must be inserted correctly. To determine the orientation, place the regulator on its back, so that the printing on the body is uppermost. From left to right, the three legs are IN, COMMON, and OUT

TESTING THE BOARD

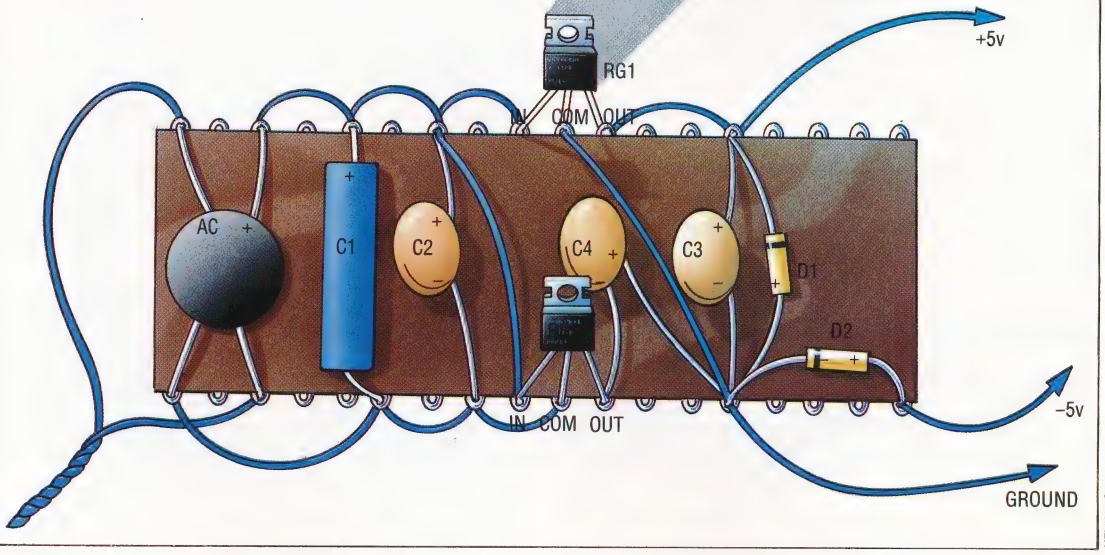
If the power supply and the power supply points to the ICs are both fine and the chips have been inserted as described above, you can now attempt to power up the board. If you haven't tied pin 9 of IC1 to the analogue ground, with an open circuit at the IN− and IN+ points, you can expect to see a few spurious digits displayed on LED1 and LED2. If these digits cycle through a repeating pattern, try moving the position of the interconnecting wiring on the board.

With IC1 pin 9 connected to the analogue ground and an open circuit on the input, you should expect to see either a 00000 display or a spurious 1 or 2 on LED1. With IC1 pin 9 grounded and IN− and IN+ shorted together, you should have a stable (non-flashing) display of 00000 or 00001. If you have had to compromise on D1 (the voltage reference diode) or C1 (the integrating capacitor) you may not get such a good initial performance. In any event, a cycling display indicates a poor wiring layout, so rearrange the wires until the display remains stable.

In the next instalment, we show how to construct the input attenuator (the unit at present can only measure voltages between 0v and 1.9999v) and discuss modifications.

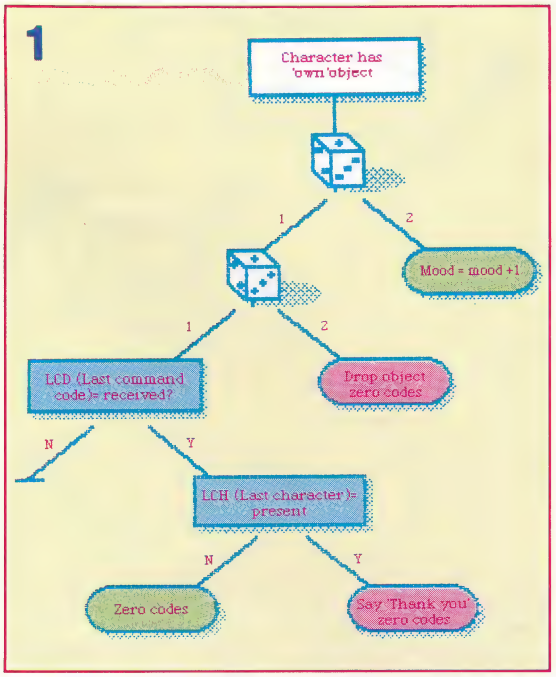


Power Supply Board



KEVIN JONES

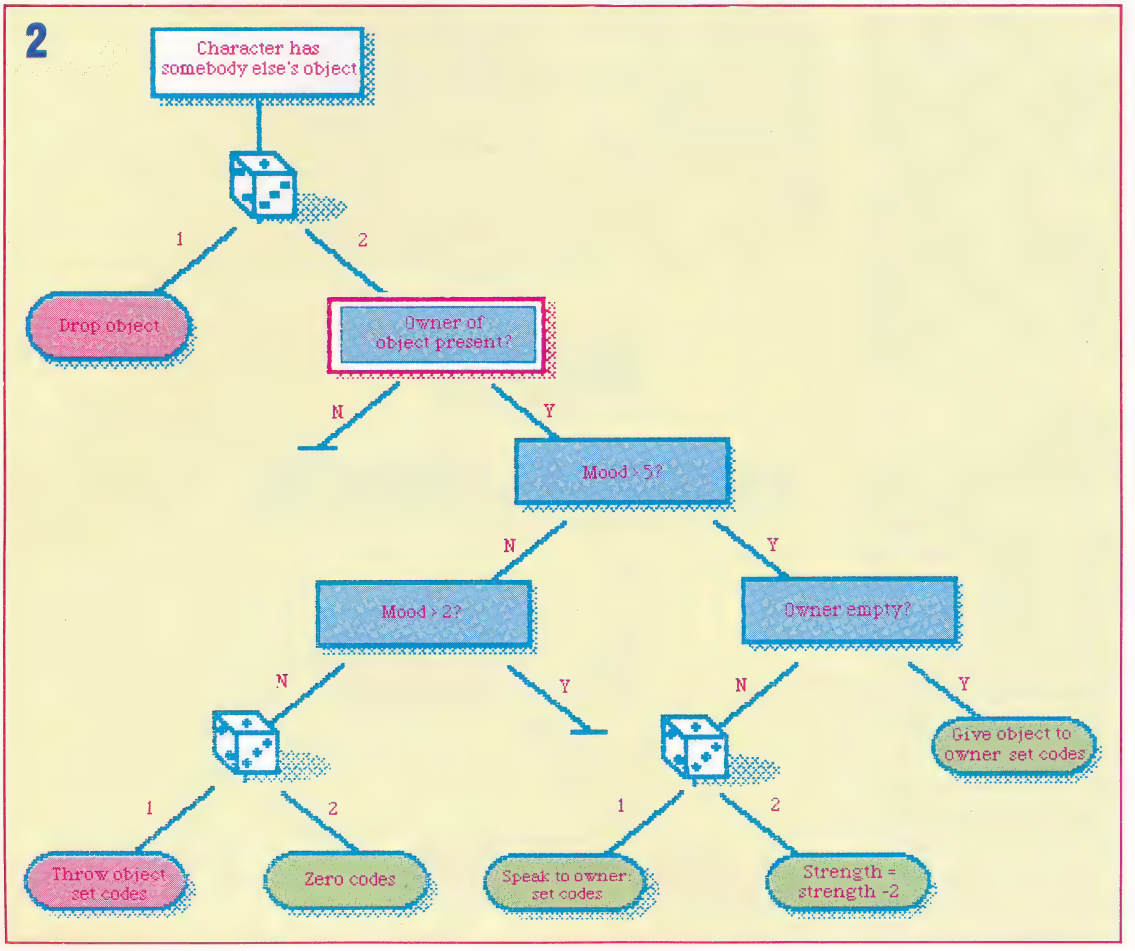
OBJECT LESSON



Although the actual number of objects that can be handled in the Dog and Bucket is not very large, the way in which they are handled is of the utmost importance. By outlining four tree structures and discussing the various implications of 'ownership', we can more fully define the requisites for programming a good adventure game.

The way in which our characters handle objects is very important in the Dog and Bucket and a large number of factors have to be taken into consideration. First of all, objects fall into two main categories: edible (the sandwich and the pasty) and drinkable, but also included are three other objects (the ashtray, stool and tin), which serve little purpose other than to add atmosphere. The tin, however, has implications for the plot, which we shall be looking at later.

The other main attribute of objects is their ownership. Certain drinks belong to certain characters, for example, and we'll obviously need



to keep track of each character's possessions. Just as importantly, we'll have to keep tabs on what each character wants. All these factors can be checked using tree structures similar to the ones we've looked at before, though rather more complicated. To make things easier, we'll tackle the question of objects in four stages.

First, let's consider the possible courses of action open to a character who is in possession of an allotted object. The first diagram shows a simple tree structure to deal with this contingency. Points to note include the dice symbols, which represent 'random nodes' whose value (which will determine the node that's branched to next) is fixed randomly when they are encountered during traversal of the tree.

As well as random nodes, there are three other node types in the diagram. The red ones are terminal nodes, at which a message will be printed and variables will be updated. The green nodes are also terminal nodes, but these will simply update variables without altering the display, so they can be dealt with separately when we enter the tree into our program. Finally, the blue nodes are simple choice nodes, each of which will hold a conditional value and branch accordingly.

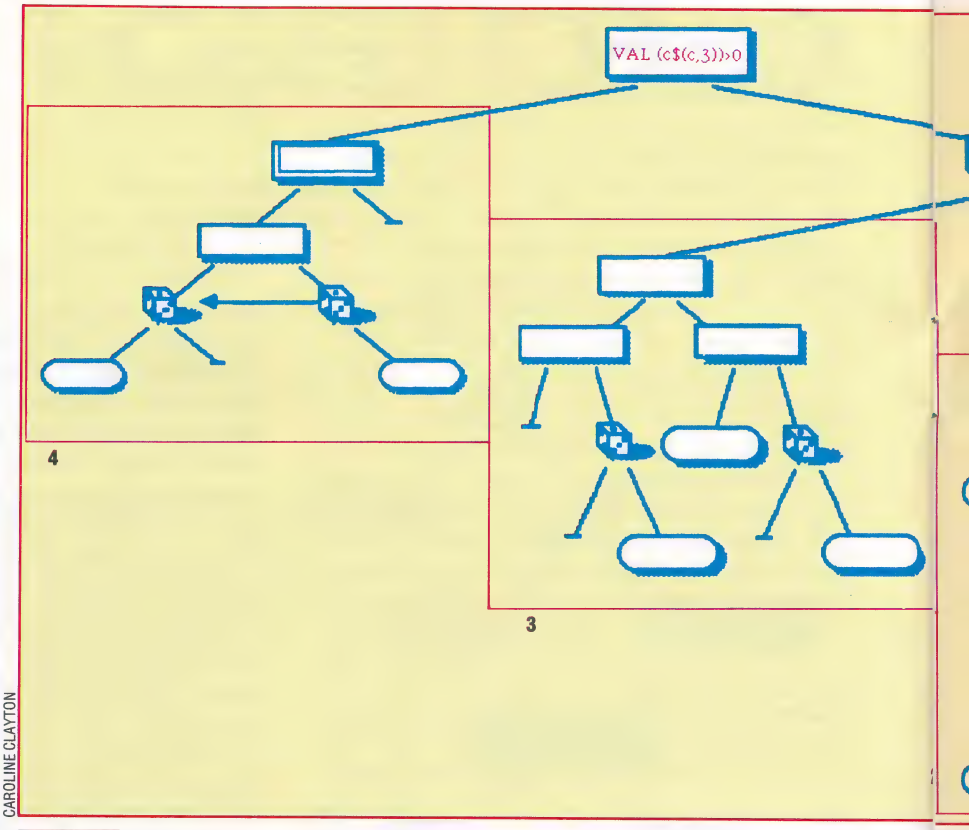
You can see from this diagram that when a character has an 'own' object, one of four outcomes, represented by the four terminal nodes, is possible. There is a random chance that the character's mood will improve, by virtue of being able to indulge in a favoured tipple. Alternatively, but less likely, the character may either drop the object or, if the object has just been received and the donor is still present, will say Thank you. Note that three of the terminal nodes result in the codes LCH and LCD being set to zero, thereby ending any previous interaction with other characters.

Structures such as these are purely arbitrary, which allows us the freedom to design quite different trees for the same purposes. With that in mind, let's move on to the second diagram, which shows what might happen when the character in question has come into possession of someone else's object.

SECOND TREE NODES

The nodes in this tree are colour-coded in a similar fashion to those in the previous tree, except for one node ringed in red. This acts in a similar way to a terminal node, so that when it is reached, the program will jump out of the tree to a routine somewhere else in the program. However, once that routine has been executed, and according to the result obtained, the program may jump back into the tree and continue descending through it.

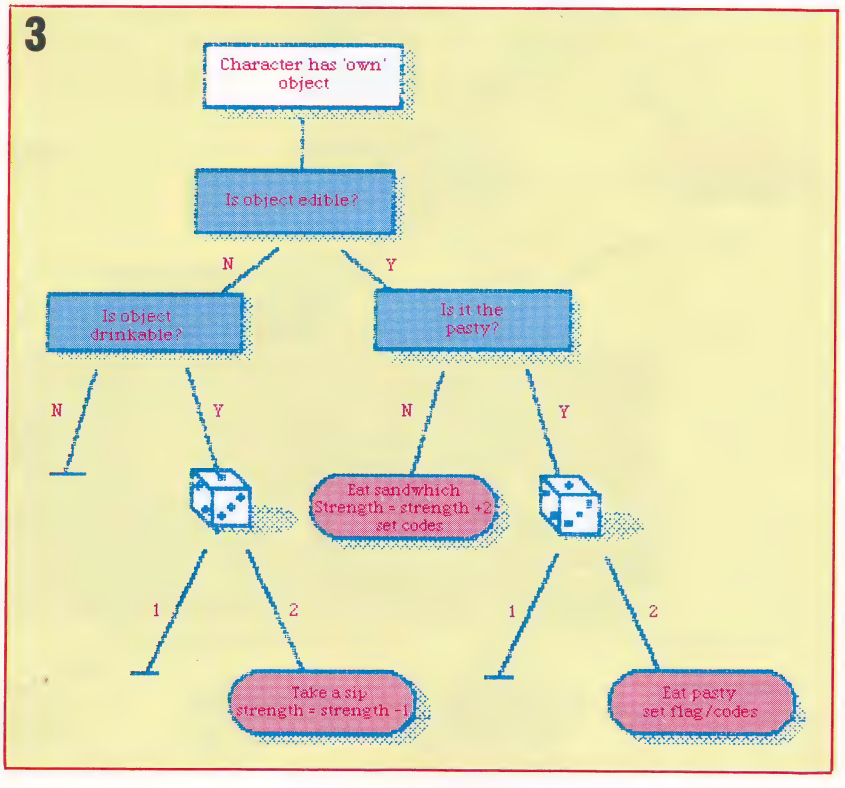
To find out if the owner of the object held by our character is in the same room, we need to run through the location attributes (c$(character,2)) of each person. At this point, the node in question pushes us out of the tree into a simple subroutine that does this for us. If the relevant character is in fact present, we'll jump back into the tree starting at the next node (testing mood>5), but if not, our

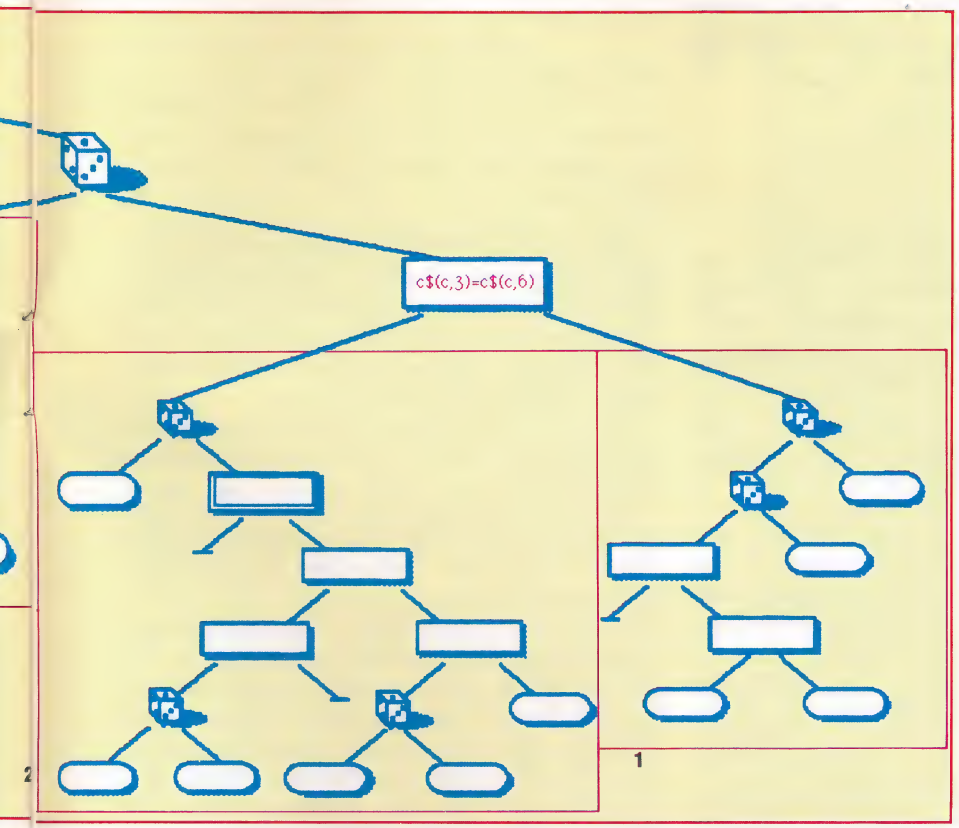


CAROLINE CLAYTON

Tree Of Trees

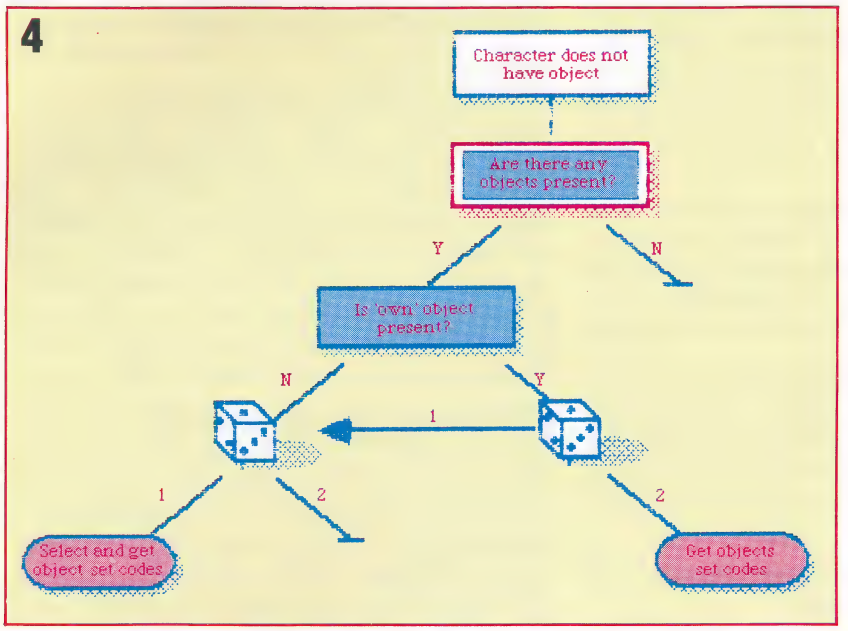
The four tree structures discussed in this instalment can all be combined to form one large tree, as shown above. The program first checks the value of the expression (VAL(c$(c,3))>0) which will evaluate to TRUE if the character in question (c) is carrying an object, and FALSE otherwise. If FALSE, the flow of control drops through the main tree to the section shown (in greater detail) in diagram **4**. If TRUE, the program makes a random decision whether to traverse the structure shown in diagram **3** (eat/drink objects) or whether to pass on and test for 'own' objects by evaluating the expression (c$(,c3)=c$(c,6)). Control then passes to one of the two structures shown in diagrams **1** and **2**





descent is abandoned and we're returned to where we began.

It's highly probable that the character will drop the object, but if this doesn't happen, the program checks for the presence of the owner. At this point, if the owner is close by, the program decides that the object should either be handed over (providing the recipient is empty-handed) or thrown at the owner, if the character is in a particularly bad mood. If the owner's hands are full, and therefore incapable of taking the object, there is a random possibility that the character may say: Excuse me, I think I must have got your drink . . ., or something of that order.



The only other thing to notice is that one of the terminal nodes involves reducing the character's strength by two points, since there is every possibility the character will take a drink from the glass if unable to return it to its original owner — and as we all know, mixing drinks is liable to have ill effects! Apart from this, however, neither of the trees just outlined makes any provision for eating or drinking. Therefore, we'll also need a structure to take care of this, which is shown in the third diagram.

This tree is colour-coded in the same way as the previous two and a character's strength can again be reduced by drinking, though not by as much since at this point there is no test for whether the right drink is being sampled. Eating the sandwich increases the character's strength by two and sets the LCD accordingly. Eating the pasty, however, sets a flag, which is for plot purposes.

Since the point has arisen, we may as well let you in on a few secrets and reveal the plot behind the Dog and Bucket scenario. You'll no doubt have pondered the significance of the cat food tins in the kitchen, and guessed that the famous Dog and Bucket Cornish pasties are, in fact, made from a well-known (but cheap) brand of pet food.

FOOD POISONING

Eating a pasty will result in the untimely demise of one or more of our characters, though exactly which ones will differ from game to game. We therefore need to set a flag to indicate which, if any, are likely to suffer from food poisoning. Whether the death of a character is treated as a mystery or a murder by the other denizens of the pub will depend on a number of other factors, which we'll look at in the next instalment. In the meantime, let's continue the examination of the object control structures.

Should a character be empty-handed, we'll use the tree outlined in the fourth diagram, in which there is another node that throws us out of the tree to take in a subroutine. This routine tests to see if there are any other objects present in the room, returning if there are and then testing whether or not the character's 'own' object is present. If it is, there is a random chance of picking it up, but if not, there's the possibility of another object being picked up.

Bear in mind that if the characters don't pick up their 'own' objects, there is a further possibility of them picking up other ones, since the node in question jumps across the tree, thereby creating this chance. This means that objects are not likely to be left lying around for long, but that characters will *tend* to pick up their own drinks.

Finally, these four trees can be put together to form one large structure, and each of the conditions that need to be tested can be expressed as a simple logical expression. Since some nodes will want to test the same conditions at different times, we'll assign the values of the conditional statements to a new array, using elements of it (where appropriate) to build up tree structures.

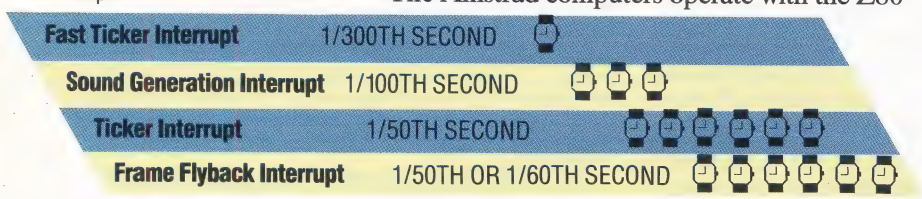
TICKED OFF

One of the most important features of any operating system is its timing. Like most other home micros, the Amstrad CPC 464/664 OS uses two methods of system timing — interrupts and events. We begin a two-part discussion showing you how to use these OS facilities to enhance your machine code programming.

The system timing governs any operation that must be carried out at a known instant — for example, flashing ink colours on the screen must be carried out during a 'frame flyback' period (the time taken to update the video display). There are two different methods used to implement this timing in the Amstrad computers; the first is controlled by the hardware and is termed an *interrupt*, and the second is controlled by the software and is called an *event*. We'll deal with the system's use of interrupts to begin our discussion.

An interrupt is a signal that is generated outside the Z80 microprocessor, which suspends execution of the current program while the signal is being dealt with.

The Amstrad computers operate with the Z80 programmed for mode 1 interrupts — that is to say whenever an interrupt is detected, control is passed to the interrupt-handling routine which resides in memory at $0038. In theory, an interrupt may be generated at any time by any device that requires attention from the Z80, although on an unexpanded Amstrad computer only one type of interrupt can occur, which is known as a *timer interrupt*.



System Interrupts
The Amstrad's processor can be interrupted by the operating system in four ways. These timer interrupts occur at regular intervals, to synchronise regular OS functions, such as keyboard scanning, updating the video display and controlling data transfer to the Amstrad's sound chip

The timer interrupt is generated directly from the system clock (running at 4MHz) by the ULA, and it occurs every 300th of a second. All internal timing (such as that required by the BASIC AFTER and EVERY commands) is a function of this timer interrupt.

The timer interrupt is handled directly by a part of the firmware called the *kernel*, which interacts with the rest of the firmware by signalling different levels of interrupt, as shown in the System Interrupts diagram. Let's consider each of these four signals separately:

- The *fast ticker interrupt* occurs whenever the kernel receives a timer interrupt.
- The *sound generation interrupt* occurs after every three timer interrupts, and is used by the firmware to synchronise data transfer to the sound chip. It is not possible for the user to detect when this interrupt occurs, except by maintaining a count of the fast ticker interrupts.
- The *ticker interrupt* occurs after every six timer interrupts and is used by the firmware to initiate a keyboard scan.
- The *frame flyback* occurs after every five or six timer interrupts — depending on the type of video display being used — and is used for any screen-dependent timing.

The firmware maintains a count of fast ticker interrupts and this is available to the user, allowing timing loops to be set up without the need for a program to suspend its activities. The count may be set by calling KL_TIME_SET, which requires the value for the timer to be passed in the HL register pair. The current count value may be determined at any time by calling KL_TIME_PLEASE, which returns the value in HL. An example program of how to use these two routines is given here; this program polls the keyboard for a specified amount of time before returning. It could be incorporated into a program that permits the user a certain response time before performing an operation (such as displaying a help message).

It is possible to use the timer interrupts directly by patching the RST 7 entry at $0038. The patched code should first copy out the existing three-byte entry, and then load in the new entry (which will normally be a jump to an address in memory). This new entry should be relocatable so that if a second routine copies it out again, it will still function correctly.

The new routine can perform whatever task is required (for example, updating a clock) and should then execute the original entry by jumping to the location in which it was saved. This procedure ensures that any existing timer interrupt functions, such as polling the keyboard, are still carried out. The idea of inserting a 'wedge' of code into an operating system interrupt routine has been met in previous instalments of the machine code series (see, for example, page 1139).

EXTERNAL INTERRUPTS

The discussion so far has assumed that any interrupt received by the kernel was generated by the ULA, but it is possible that an interrupt may be generated by an external peripheral. For example, a serial interface could generate an interrupt whenever a character becomes available — a

routine would, in this case, have to be provided to read the character from the interface.

For this to be possible the kernel must be able to distinguish between a timer interrupt and an external interrupt. Normally, when an interrupt is detected, the Z80 is disabled from receiving any more interrupts — this is to ensure that any timer interrupts are not violated. The kernel enables interrupts from within the interrupt-handling routine, and if the interrupt signal to the Z80 is still low (indicating an interrupt request) then a second interrupt will be generated. As a timer interrupt is not duplicated at this stage, this can be used to detect an external interrupt provided that the interrupt signal remains low until it has been serviced. When an external interrupt is detected, control is passed to $003B, and so this location should be patched with a jump to the interrupt-servicing routine — which then services the requesting device.

The system hardware supports *non-maskable interrupts* (NMI) as well as standard interrupts. The NMI is different only in that it cannot be disabled at any time, and its use is not recommended as it will affect system activities with stringent timing requirements. In particular, the cassette and disk handling sections of the firmware — and parts of the sound-handling sections — continue to operate with interrupts disabled, but would be seriously affected if NMIs were used.

SOFTWARE EVENTS

At the simplest level *events* may be described as the software equivalent of interrupts — particular tasks are performed separately from any main program by the firmware keeping a count of what events are to be processed and when.

Although events may be used to handle external interrupts, they are most useful for complicated programs involving simulation, or that require operation independent of a main foreground program. For example, the Amstrad SSA-1 speech synthesiser uses an event to allow speech to be generated transparently from within a BASIC program.

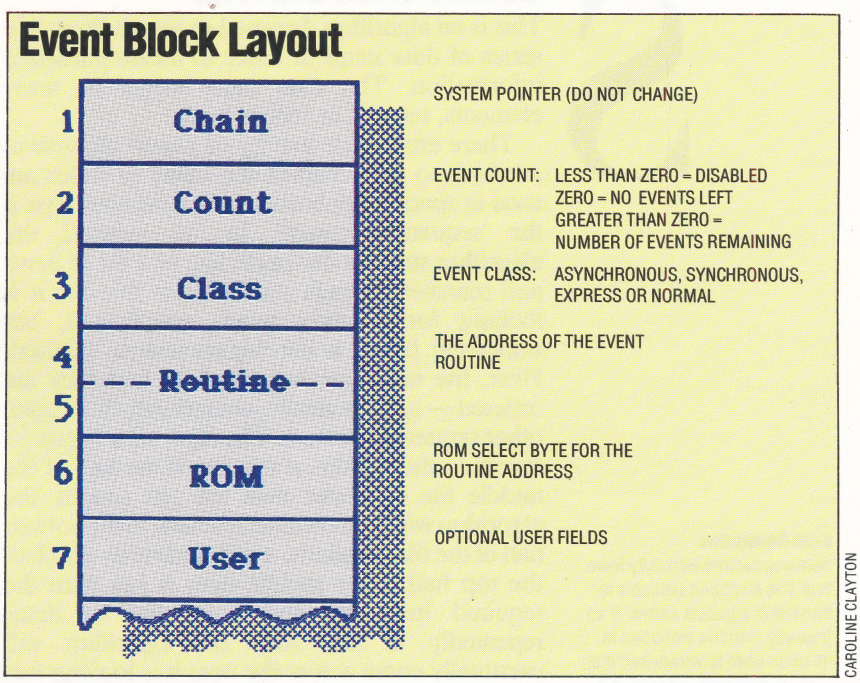
Events are presented to the operating system by an *event block*. This is simply a block of seven contiguous bytes located somewhere in the central 32 Kbytes of memory. An event block has the layout shown in our diagram; held in the block are three fields, essential to all events, that characterise the type of event and where it will be processed.

Within the event block is the *event count*, which keeps track of how many times an event has occurred. The process of incrementing the event count is known as *kicking*. Whenever an event is kicked the count is stepped up be one; after an event has been processed by calling the event routine, the count is decremented by one. Any event may be disarmed by giving the count a negative value.

The *event class* indicates whether it is *synchronous* or *asynchronous* in relation to the main program. Asynchronous events are processed independently of the main program and are intended for applications that require almost immediate action.

Synchronous events are placed in a queue when they occur, so that the main program can process them when necessary. Synchronous events are processed in the order they occur. In addition, the event may also be *express* or *normal*.

In the next instalment, we'll consider the differences between the various classes of events, and the way in which the operating system deals with them. It may appear that using events presents a number of unnecessary complications, but in fact the event structure on the Amstrad is both powerful and flexible, and has the added advantage for the machine code programmer of being entirely system compatible.



CAROLINE CLAYTON

The Main Event
Events are the software equivalent of interrupts and the event block is used by the operating system to keep track of any events set up. Seven consecutive bytes are used to hold the address of the event routine to which the block relates, the number — if any — of events that remain to be processed, and the class of event

```
; Example of using the Kernel timer routines
;
;
SET.TIME:       EQU   £BD10       ; read the fast tick time count
GET.TIME:       EQU   £BD0D       ; set the fast tick time count
;
READ.CHAR:      EQU   £BB09       ; scan keyboard for char
PAUSE:          EQU   300         ; Pause for 300 ticks
;
                LD    HL,0        ; set time to zero
                LD    DE,0
                CALL  SET.TIME
                LD    BC, PAUSE   ; set wait period
;
; wait for a character to become available from
; the keyboard
;
WAIT:           CALL  READ.CHAR   ; character ready?
                RET   C           ; if so go back
;                                   with carry set
                CALL  GET.TIME    ; find the time
                SBC   HL,BC       ; one second?
;                                   (or greater)
                JR    C,WAIT      ; if not then retry
;
; if we get here, then routine has timed out
; go back with carry false to flag a timeout
;
                RET
;
```

SEARCH ALGORITHM

This is an algorithm designed to search through a series of data items in order to locate particular information. The data items might be array elements, records or record keys.

There are a wide variety of *search algorithms* available to the programmer, many of which are used in specific applications. The simplest type is the 'sequential search'. In this method, the algorithm starts at the beginning of a list of items and searches through until it finds the one it is looking for. Slightly more complicated, but somewhat faster, is the 'binary search' method. Here, the items are organised so that they are ordered — alphabetically, numerically or in some other sequential pattern. The algorithm begins its search in the middle of the file. If the key of the middle file is greater than the one sought, the algorithm will then confine its search to the bottom half of the file. Similarly, it will confine its search to the top half if the middle item is less than the required item. By slicing the list of items repeatedly in this way, the algorithm will eventually come across the item it is looking for.

'Breadth-first' and 'depth-first' techniques are similar in their methods. In breadth-first search algorithms, the information is organised into a 'tree' system, and the algorithm begins its search by examining a particular node level. Once these have been exhausted, the search then proceeds to another level. Conversely, depth-first algorithms start at a particular node and search all the paths leading from that node. If the sought-after item is not found, the algorithm moves back along the path until it finds a node that has not been searched, searches all the paths from there and so on. (For more information on breadth-first and depth-first search techniques, refer to our series on AI beginning on page 1221.)

SECTOR

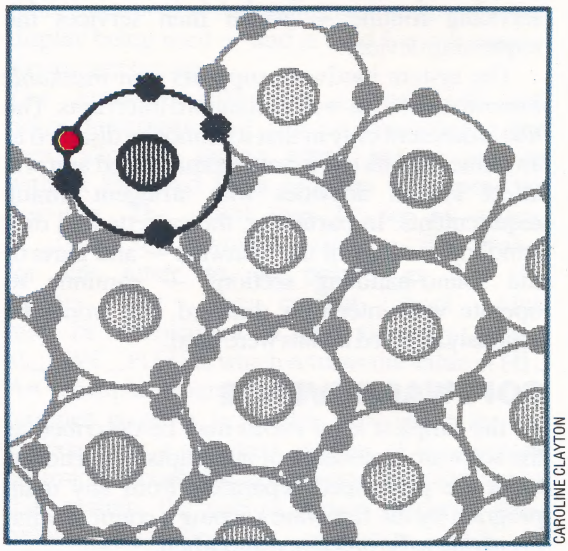
This is a subdivided area of a track on a disk, often corresponding to a physical record, on which data is stored (see page 124). In order for the computer's disk operating system to be able to find the information stored on *sectors*, each has its own address, which is made up of the track number and the position of the sector on that track.

There are two types of sectors used with disks. 'Hard sectors' are actually built onto the disks themselves, usually in the form of small holes drilled into the disks. Thus the sector can be physically detected by the disk drive mechanism. On the other hand, 'soft sectors' are set up magnetically — usually when the disk is formatted. This is performed by the software within the disk drive or computer, which instructs the read/write head (see page 1460) of the drive to place magnetic markers (known as 'sync-marks') at regular intervals on the tracks.

SEMICONDUCTOR

This is a chemical element, such as silicon or gallium, whose conducting properties are dependent upon their temperature. Thus, at one temperature, they will not conduct at all, but will become good conductors of electricity if the temperature is raised. The reason for this is that the electrons in the outer layers of the atom are dislodged from their orbits when heat is applied to them. This, therefore, generates a number of 'free' electrons within the material and, of course, an equal number of 'holes' in the atoms where the electrons have been dislodged. By applying a potential difference to either side of the *semiconductor* material, the electrons can be moved from one side to the other, thereby conducting an electrical current.

Semiconductor materials have four potentially free electrons in their outermost shell. We can alter the properties of these materials by introducing other elements — a process known as 'doping'.



CAROLINE CLAYTON

Semi-Conductive
Semiconductors normally have four free electrons available to bond with adjacent atoms. If an impurity with five electrons in its outer shell is introduced then surplus electrons are available to move through the material when a potential difference is applied

Introducing an element to a semiconductor such as arsenic, which has only three 'free' electrons, creates a surplus of positive holes and is therefore known as a 'p-type semiconductor'. On the other hand, if we introduce an element with five free electrons in its outer shell, there will be a corresponding increase in free electrons, thus creating an 'n-type semiconductor'. By placing a p-type and n-type semiconductor together, electrons can be made to flow from one to the other, as the electrons attempt to flow from the negative to the positive side. This effect is the basis of the transistor.

SEQUENTIAL ACCESS

This is a method of obtaining data whereby the order of accessing remains the same, regardless of the data you are trying to obtain. Furthermore, unless gaps are left between the data for extra blocks of information, new data will have to be added at the end of previous blocks.

By far the most common medium for the *sequential access* of data is magnetic tape. However, many floppy disk operating systems also place their files in sequential order during file organisation.

Snowball, from Level 9 Computing, was the first popular adventure game for the Amstrad CPC 464/664, as well as the first part of the Silicon Dream trilogy. Jon Kaye looks at the follow-up, Return to Eden, and finds that little has changed

THE SNOWBALL EFFECT

Welcome to "Return to Eden"
Copyright (C) 1984 Level 9
Computing.
You are in the control room of
a crashed stratoglider. A
vision screen shows the strange
plants of Eden. There's an
opening behind you.

In Return to Eden, you have been set up to take the blame for the sabotage of the Snowball 9 colonising spacecraft, which happened in the first game. After being sentenced to death, you have stealthily managed to escape to the planet Eden where the action takes place.

After landing, you must try and make your way to the robot city, the only 'civilised' place as far as you know, and your only hope of salvation. Between here and there, however, are the wicked indigenous inhabitants as well as the entirely unpredictable wildlife — you can be killed instantly by a river rat, for example. This sparse information is all that elevates you from abject helplessness.

The game is billed as having some 240 separate screens, and so it might be too much to expect all of these to be exquisitely drawn. It would have been nice, however, if just a handful of the screens were detailed a bit more than the present, almost sketchily drawn, screen displays. But the constant change in scenery is nevertheless an attractive and engaging feature of the game.

If your nerves can handle it, there are rarely any moments at which you can feel safe, or even relaxed. Whichever way you turn, there are dangers either to be confronted or, more usually, avoided. Besides the perils of the wilderness and wildlife, there are the inhabitants of the robot city itself, who recognise anything outside their domed city as enemies to be destroyed. It is in fact this very point that provides you with a positive purpose to your quest — to prevent the robots from destroying the Snowball 9, the very ship that wants you dead.

After the first hour or two, you'll realise that this is definitely not an easy game to complete. Its very nature requires an intuitiveness and elusiveness usually associated with master criminals. Above all, it requires imaginative probing into the unknown. However, the characters that confront you seem just as devious, somehow always finding you regardless of your strategies.

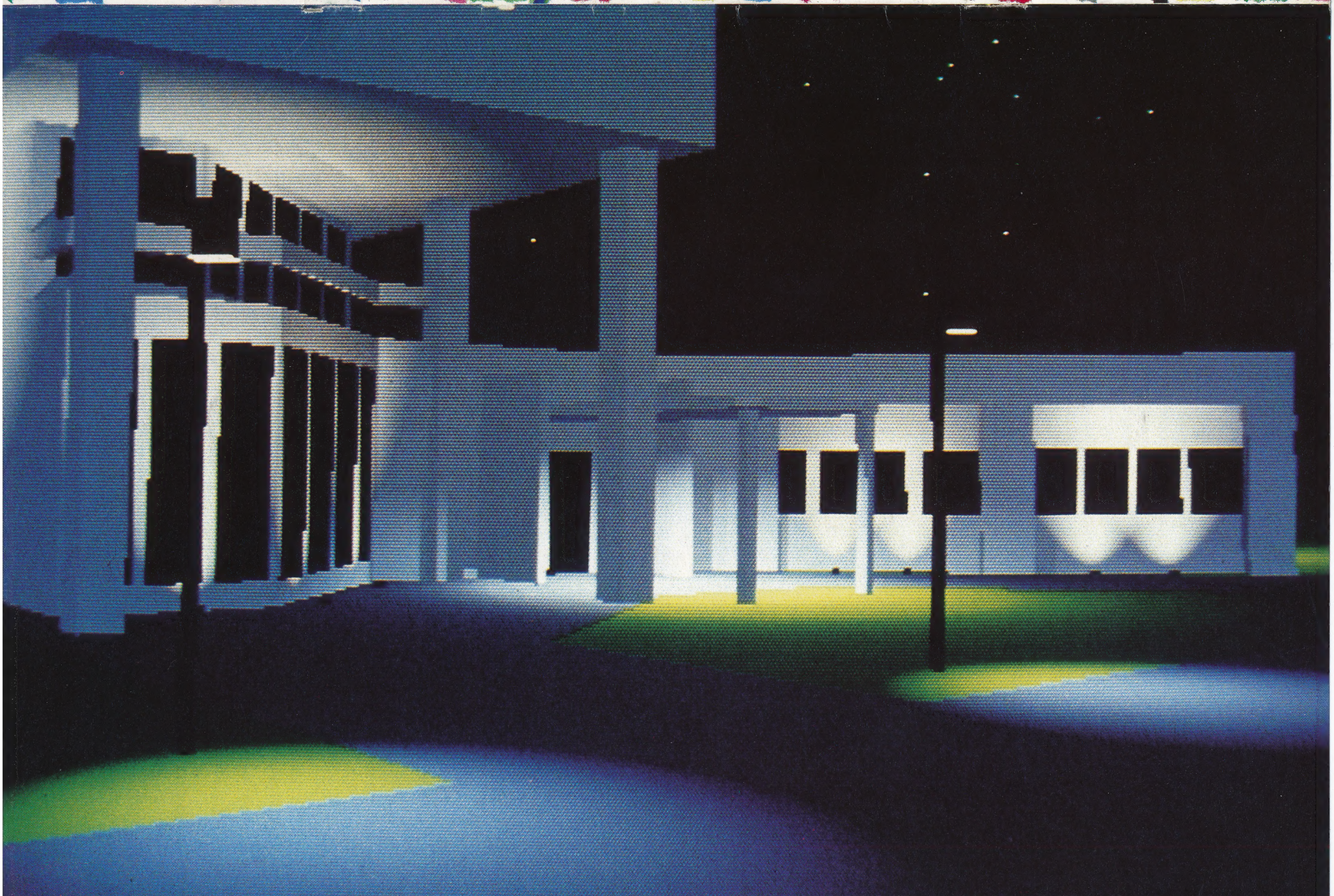
As with any adventure worth its salt, there are no blatant clues to help you along. Ultimately, of course, your success will depend on the information you can glean from the text, but the game does give very little away. If your powers of endurance are lacking, you can always send an SAE to the publishers requesting a few hints — a last, but by no means honourable, resort.

Without giving anything away, however, there are certain aspects of Return to Eden that you should keep in mind. By examining thoroughly your surroundings and keeping notes on the attributes and weaknesses you've already encountered, it's possible to generate a rough idea of the logic used throughout. By taking this into consideration, the chances of making a successful move on the first try after a prompt seems to greatly increase. In other words, there are definite patterns of behaviour, however tenuously extrapolated, that can be taken advantage of. In this sense, then, you can gain an ally in an entirely alien world.

Being an adventure game, you would expect endless possibilities as to what may or may not happen to you, or what you can do — but this is just not so. At each prompt, you often have many choices and can just as often type in any short phrase you wish. But so often you are answered with: 'You can't do that', or 'You can't go that way', meaning that you can only go one way and ultimately do only one thing. This might work better if the scenario was more along the lines of a scientist searching for a panacea.

You may find, therefore, that your Return to Eden could be postponed for a while, since there are certainly other more tempting apples about. But then again, we all have our own ideas of what paradise should be. There will be many awaiting the final instalment in this saga — The Worm in Paradise — due for release soon.

The game can be obtained from: Level 9 Computing, 229 Hughenden Road, High Wycombe, Bucks.



© 1983 NAKAMAE, NISHITA, OHBAYASHI—
HIROSHIMA UNIV.